10

Classes: A Deeper Look, Part 2

But what, to serve our private ends,
Forbids the cheating of our friends?
—Charles Churchill

Instead of this absurd division into sexes they ought to class people as static and dynamic.
—Evelyn Waugh

Have no friends not equal to yourself.
—Confucius

OBJECTIVES

In this chapter you'll learn:

- To specify `const` (constant) objects and `const` member functions.
- To create objects composed of other objects.
- To use `friend` functions and `friend` classes.
- To use the `this` pointer.
- To use `static` data members and member functions.
- The concept of a container class.
- The notion of iterator classes that walk through the elements of container classes.
- To use proxy classes to hide implementation details from a class's clients.
- To specify `const` (constant) objects and `const` member functions.

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, 25, 26, 27, 28	
Programming Output	29, 30, 31	
Correct the Code	32, 33, 34, 35, 36	
Lab Exercises		
Lab Exercise 1 — Simple Calculator	YES NO	
Follow-Up Questions and Activities	1, 2, 3, 4	
Lab Exercise 2 — Integer Set	YES NO	
Follow-Up Question and Activity	1	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2, 3, 4, 5, 6	
Programming Challenges	1, 2	

Prelab Activities

Matching

Name: ____________________ **Date:** ____________________

Section: ____________________

After reading Chapter 10 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. Free store or heap	a) Used when only one copy of a variable should be shared by all instances of a class.
___	2. `new` and `delete`	b) Describing functionality of a class independent of its implementation.
___	3. Iterator	c) An object that "walks through" a collection.
___	4. Memory leak	d) A region of memory for storing objects created at execution time.
___	5. Data abstraction	e) Data structure in which the last item inserted is the first one removed.
___	6. `friend` function	f) Defined outside the class's scope, yet has access to `private` members of the class.
___	7. `this` pointer	g) Data structure in which the first item inserted is the first one removed.
___	8. LIFO	h) Operators used for performing dynamic memory allocation and deallocation.
___	9. FIFO	i) An implicit argument to all non-`static` member-function calls.
___	10. `static` class variable	j) Occurs when objects are allocated but never deallocated.

Prelab Activities

Name:

Fill in the Blank

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

Fill in the blanks in each of the following statements:

11. Classes can be composed of ___________ of other classes.

12. Keyword ___________ specifies that an object is not modifiable.

13. A `const` data member must be initialized in the ___________.

14. If a member initializer is not provided for a member object, the member object's ___________ is called.

15. The ___________ pointer references both the non-`static` member functions and non-`static` data members of the object.

16. The ___________ operator allocates space for an object, runs the object's constructor and returns a pointer of the correct type.

17. To destroy a dynamically allocated object, the ___________ operator is used.

18. A(n) ___________ data member represents "class-wide" information.

19. ___________ are known as last-in, first-out (LIFO) data structures; ___________ are known as first-in, first-out (FIFO) data structures.

20. ___________ are designed to hold collections of objects.

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 the purpose of declaring certain objects with keyword `const`?

22. What are `friend` functions? Where in a class definition is a `friend` function specified?

23. What are `static` data members? Why might they be used? What is their scope?

24. What is a stack?

Prelab Activities

Name:

Short Answer

25. What is a queue?

26. What is a proxy class? Why might it be used?

27. What features does C++ provide for dynamic memory allocation?

28. What is meant by "cascading function calls?" How is such cascading accomplished in designing a class?

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.]

29. What is output by the following code? Use the `Increment` class defined in Fig. 10.4 and Fig. 10.5.

```
Increment object( 65, 7 );

cout << "Before incrementing: ";
object.print();

for ( int j = 0; j < 6; j++ ) {
   object.addIncrement();
   cout << "After increment " << j + 1 << ": ";
   object.print();
} // end for
```

Your answer:

30. What is output by the following code? Assume the use of the `Employee` and `Date` classes defined in Fig. 10.10–Fig. 10.13.

```
Date d1( 2, 14, 1963 );
Date d2( 1, 29, 2001 );
Employee e( "John", "Doe", d1, d2 );

cout << '\n';
e.print();
```

Prelab Activities

Name:

Programming Output

Your answer:

31. What is output by of the following program?

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

// class Test definition
class Test
{
public:
   Test( int = 0 );
   void print() const;
private:
   int x;
}; // end class Test

// default constructor
Test::Test( int a )
{
   x = a;
} // end class Test constructor

// function print definition
void Test::print() const
{
   cout << x
        << this->x
        << ( *this ).x << endl;
} // end function print

int main()
{
   Test testObject( 4 );

   testObject.print();
} // 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 or 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.]

32. The following defines class `Increment`. (Note the use of `const` data member `increment`.):

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

// class Increment definition
class Increment
{
public:
   Increment( int c = 0, int i = 1 );
   void addIncrement() { count += increment; }
   void print() const;
private:
   int count;
   const int increment;
}; // end class Increment

// constructor
Increment::Increment( int c, int i )
{
   count = c;
   increment = i;
} // end class Increment constructor

// function print definition
void Increment::print() const
{
   cout << "count = " << count
        << ", increment = " << increment << endl;
} // end function print
```

Prelab Activities

Name:

Correct the Code

Your answer:

33. The code that follows is a definition for class `Time` and its member functions.

```cpp
// class Time definition
class Time
{
public:
   Time( int = 0, int = 0, int = 0 );

   void setTime( int, int, int ) const;

   void setHour( int ) const;
   int getHour() const;

   void setMinute( int ) const;
   int getMinute() const;

   void setSecond( int ) const;
   int getSecond() const;

   void printUniversal() const;
   void printStandard();
private:
   int hour;
   int minute;
   int second;
}; // end class Time
```

Prelab Activities

Name:

Correct the Code

```cpp
// Member function definitions for Time class.
#include <iostream>
using namespace std;

#include "time.h"

// constructor function to initialize private data
// default values are 0 (see class definition)
Time::Time( int hr, int min, int sec )
{
   setTime( hr, min, sec );
} // end class Time constructor

// set values of hour, minute and second.
void Time::setTime( int h, int m, int s )
{
   setHour( h );
   setMinute( m );
   setSecond( s );
} // end function setTime

// set hour value
void Time::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0;
} // end function setHour

// set minute value
void Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0;
} // end function setMinute

// set second value
void Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ? s : 0;
} // end function setSecond

// get hour value
int Time::getHour() const
{
   return hour;
} // end functiongetHour

// get minute value
int Time::getMinute() const
{
   return minute;
} // end function setMinute

// get second value
int Time::getSecond() const
{
   return second;
} // end function getSecond

```

Prelab Activities

Name:

Correct the Code

```
// display universal format time: HH:MM
void Time::printUniversal() const
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
} // end function printUniversal

// display standard format time: HH:MM:SS AM (or PM)
void Time::printStandard()
{
   cout << ( ( hour == 12 ) ? 12 : hour % 12 ) << ":"
        << ( minute < 10 ? "0" : "" ) << minute << ":"
        << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

Your answer:

34. The code that follows is a definition for class `Time`. Note the member function that begins a new day by resetting the `hour` to zero.

```
// class Time definition
class Time
{
public:
   Time( int = 0, int = 0, int = 0 );

   void setTime( int, int, int );

   void setHour( int );
   int getHour() const;

   void setMinute( int );
   int getMinute() const;

   void setSecond( int );
   int getSecond() const;

   // function newDay definition
   void newDay() const
   {
      setHour( 0 );
   } // end function newDay

   void printUniversal() const;
   void printStandard();
```

Prelab Activities

Name:

Correct the Code

```
private:
   int hour;
   int minute;
   int second;
}; // end class Time
```

Your answer:

35. The following is a definition for class `Time`:

```
// class Time definition
class Time
{
public:
   Time( int = 0, int = 0, int = 0 ) const;

   void setTime( int, int, int );

   void setHour( int );
   int getHour() const;

   void setMinute( int );
   int getMinute() const;

   void setSecond( int );
   int getSecond() const;

   void printUniversal() const;
   void printStandard();
private:
   int hour;
   int minute;
   int second;
}; // end class Time
```

Prelab Activities

Name:

Correct the Code

Your answer:

36. The code that follows is a definition of the `getCount` member function. Variable `count` is a `static int` that stores the number of objects instantiated. Assume that this definition is located within a class definition.

```
static int getCount()
{
   return this->count;
}
```

Your answer:

Lab Exercises

Lab Exercise 1 — Simple Calculator

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 10.1 – Fig. L 10.3)
5. Problem-Solving Tip
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 tip 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 10 of *C++ How To Program, Seventh Edition*. In this lab, you will practice:

- Using classes to create a data type `SimpleCalculator` capable of performing arithmetic operations.
- Creating `const` member functions to enforce the principle of least privilege.

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

- Using constructors to specify initial values for data members of a programmer-defined class.

Description of the Problem

Write a `SimpleCalculator` class that has `public` methods for adding, subtracting, multiplying and dividing two `double`s. A sample call is as follows:

```
double answer = sc.add( a, b );
```

Object `sc` is of type `SimpleCalculator`. Member function `add` returns the result of adding its two arguments.

Sample Output

```
The value of a is: 10
The value of b is: 20

Adding a and b yields 30
Subtracting b from a yields -10
Multiplying a by b yields 200
Dividing a by b yields 0.5
```

Lab Exercises

Name:

Lab Exercise 1 — Simple Calculator

Template

```
// Lab Exercise 1: SimpleCalculator.h

// class SimpleCalculator definition
class SimpleCalculator
{
public:
   /* Write prototype for add member function */
   double subtract( double, double ) const;
   double multiply( double, double ) const;
   /* Write prototype for divide member function */

}; // end class SimpleCalculator
```

Fig. L 10.1 | Contents of `SimpleCalculator.h`.

```
// Lab Exercise 1: SimpleCalculator.cpp

#include "SimpleCalculator.h"

/* Write definition for add member function */

// function subtract definition
double SimpleCalculator::subtract( double a, double b ) const
{
   return a - b;

} // end function subtract

// function multiply definition
double SimpleCalculator::multiply( double a, double b ) const
{
   return a * b;

} // end function multiply

/* Write definition for divide member function */

```

Fig. L 10.2 | Contents of `SimpleCalculator.cpp`.

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

#include "SimpleCalculator.h"

int main()
{
   double a = 10.0;
   double b = 20.0;

```

Fig. L 10.3 | Contents of `CalcTest.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 1 — Simple Calculator

```
    /* Instantiate an object of type SimpleCalculator */
    cout << "The value of a is: " << a << "\n"
        << "The value of b is: " << b << "\n\n";

    /* Write a line that adds a and b through your SimpleCalculator
       object; assign the result to a variable named addition */
    cout << "Adding a and b yields " << addition << "\n";

    double subtraction = sc.subtract( a, b );
    cout << "Subtracting b from a yields" << subtraction << "\n";

    double multiplication = sc.multiply( a, b );
    cout << "Multiplying a by b yields " << multiplication << "\n";

    /* Write a line that divides a and b through the
       SimpleCalculator object; assign the result to a
       variable named division */
    cout << "Dividing a by b yields " << division << endl;
} // end main
```

Fig. L 10.3 | Contents of `CalcTest.cpp`. (Part 2 of 2.)

Problem-Solving Tip

1. All of `SimpleCalculator`'s member functions should have return type `double`.

Follow-Up Questions and Activities

1. Why doesn't the `SimpleCalculator` class have a constructor?

2. Why are no `private` data members needed for class `SimpleCalculator`?

Lab Exercises

Name:

Lab Exercise 1 — Simple Calculator

3. Modify your class so that `SimpleCalculator` has a `private` data member called `answer`. After performing an operation, assign the result to `answer`. Add a member function named `getAnswer` to retrieve the result of the last arithmetic operation performed by the object. Also, add a constructor for class `SimpleCalculator` that initializes the value of `answer` to `0`.

Lab Exercises

Name:

Lab Exercise 1 — Simple Calculator

4. Modify the program so that the `SimpleCalculator` class has an `input` member function that allows the user to input two `double`s. The function should then store the values that were input in `private` data members. Use these two values for each of the arithmetic calculations. Create two constructors for this class, one that takes no arguments and initializes `a` and `b` to `0` and another that takes two `double`s and initializes `a` and `b` to those values. Finally, create a member function `printValues` that displays the values of `a` and `b`. A segment of the driver program might now look like this:

```
SimpleCalculator sc;   // instantiate object

sc.input();
sc.printValues();
cout << "Adding a and b yields " << sc.add() << "\n";
```

Lab Exercises

Name:

Lab Exercise 2 — Integer Set

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 10.4 – Fig. L 10.6)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

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 question. 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 10 of *C++ How To Program, Seventh Edition*. In this lab, you will practice:

- Using classes to create a data type, `IntegerSet`, capable of storing a set of integers.
- Using dynamic memory allocation with the `new` and `delete` operators.

In the follow-up question and activity you also will practice:

- Using destructors to deallocate memory that was dynamically allocated.

Description of the Problem

Create class `IntegerSet` for which each object can hold integers in the range 0 through 100. A set is represented internally as an array of ones and zeros. Array element `a[ i ]` is 1 if integer *i* is in the set. Array element `a[ j ]` is 0 if integer *j* is not in the set. The default constructor initializes a set to the so-called "empty-set," i.e., a set whose array representation contains all zeros.

Provide member functions for the common set operations. For example, a `unionOfSets` member function (already provided) creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third array's is set to `1` if that element is `1` in either or both of the existing sets, and an element of the third set's array is set to 0 if that element is 0 in each of the existing sets).

Provide an `intersectionOfSets` member function which creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set's array is set to `0` if that element is `0` in either or both of the existing sets, and an element of the third set's array is set to `1` if that element is `1` in each of the existing sets).

An `insertElement` member function (already provided) inserts a new integer *k* into a set (by setting `a[ k ]` to `1`). Provide a `deleteElement` member function that deletes integer *m* (by setting `a[ m ]` to `0`).

A `printSet` member function (already provided) prints a set as a list of numbers separated by spaces. Print only those elements which are present in the set (i.e., their position in the array has a value of `1`). Print `---` for an empty set.

Lab Exercises

Name:

Lab Exercise 2 — Integer Set

Provide an `isEqualTo` member function that determines whether two sets are equal.

Provide an additional constructor that receives an array of integers and the size of that array and uses the array to initialize a set object.

Now write a driver program to test your `IntegerSet` class. Instantiate several `IntegerSet` objects. Test that all your member functions work properly.

Sample Output

```
Enter set A:
Enter an element (-1 to end): 45
Enter an element (-1 to end): 76
Enter an element (-1 to end): 34
Enter an element (-1 to end): 6
Enter an element (-1 to end): -1
Entry complete

Enter set B:
Enter an element (-1 to end): 34
Enter an element (-1 to end): 8
Enter an element (-1 to end): 93
Enter an element (-1 to end): 45
Enter an element (-1 to end): -1
Entry complete

Union of A and B is:
{   6   8  34  45  76  93   }
Intersection of A and B is:
{  34  45   }
Set A is not equal to set B

Inserting 77 into set A...
Set A is now:
{   6  34  45  76  77   }

Deleting 77 from set A...
Set A is now:
{   6  34  45  76   }
Invalid insert attempted!
Invalid insert attempted!

Set e is:
{   1   2   9  25  45  67  99 100   }
c
```

Template

```cpp
// Lab 2: IntegerSet.h
// Header file for class IntegerSet
#ifndef INTEGER_SET_H
#define INTEGER_SET_H

class IntegerSet
{
public:
   // default constructor
   IntegerSet()
   {
```

Fig. L 10.4 | Contents of `IntegerSet.h`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Integer Set

```
      /* Write call to emptySet */
   } // end IntegerSet constructor

   IntegerSet( int [], int ); // constructor that takes an initial set
   IntegerSet unionOfSets( const IntegerSet& );
   /* Write a member funcion prototype for intersectionOfSets */
   void emptySet(); // set all elements of set to 0
   void inputSet(); // read values from user
   void insertElement( int );
   /* Write a member function prototype for deleteElement */
   void printSet() const
   /* Write a member function prototype for isEqualTo */
private:
   int set[ 101 ]; // range of 0 - 100

   // determines a valid entry to the set
   int validEntry( int x ) const
   {
      return ( x >= 0 && x <= 100 );
   } // end function validEntry
}; // end class IntegerSet

#endif
```

Fig. L 10.4 | Contents of `integerset.h`. (Part 2 of 2.)

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

/* Write include directive for IntegerSet.h here */

// constructor creates a set from array of integers
IntegerSet::IntegerSet( int array[], int size)
{
   emptySet();

   for ( int i = 0; i < size; i++ )
      insertElement( array[ i ] );
} // end IntegerSet constructor

/* Write a definition for emptySet */

// input a set from the user
void IntegerSet::inputSet()
{
   int number;

   do
   {
      cout << "Enter an element (-1 to end): ";
      cin >> number;

```

Fig. L 10.5 | Contents of `integerset.cpp`. (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Integer Set

```
         if ( validEntry( number ) )
            set[ number ] = 1;
         else if ( number != -1 )
            cerr << "Invalid Element\n";
      } while ( number != -1 ); // end do...while

      cout << "Entry complete\n";
   } // end function inputSet

   // prints the set to the output stream
   void IntegerSet::printSet() const
   {
      int x = 1;
      bool empty = true; // assume set is empty

      cout << '{';

      for (int u = 0; u < 101; u++ )
      {
         if ( set[ u ] )
         {
            cout << setw( 4 ) << u << ( x % 10 == 0 ? "\n" : "" );
            empty = false; // set is not empty
            ++x;
         } // end if
      } // end for

      if ( empty )
         cout << setw( 4 ) << "---"; // display an empty set

      cout << setw( 4 ) << "}" << '\n';
   } // end function printSet

   // returns the union of two sets
   IntegerSet IntegerSet::unionOfSets( const IntegerSet &r )
   {
      IntegerSet temp;

      // if element is in either set, add to temporary set
      for ( int n = 0; n < 101; n++ )
         if ( set[ n ] == 1 || r.set[ n ] == 1 )
            temp.set[ n ] = 1;

      return temp;
   } // end function unionOfSets

   /* Write definition for intersectionOfSets */

   // insert a new integer into this set
   void IntegerSet::insertElement( int k )
   {
      if ( validEntry( k ) )
         set[ k ] = 1;
      else
         cerr << "Invalid insert attempted!\n";
   } // end function insertElement
```

Fig. L 10.5 | Contents of integerset.cpp. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Integer Set

```

/* Write definition for deleteElement */

/* Write definition for isEqualTo */

// determines if two sets are equal
bool IntegerSet::isEqualTo( const IntegerSet &r ) const
{
   for ( int v = 0; v < 101; v++ )
      if ( set[ v ] != r.set[ v ] )
         return false; // sets are not-equal

   return true; // sets are equal
} // end function isEqualTo
```

Fig. L 10.5 | Contents of `integerset.cpp`. (Part 3 of 3.)

```
// Lab 2: SetTest.cpp
// Driver program for class IntegerSet.
#include <iostream>
using namespace std;

#include "IntegerSet.h" // IntegerSet class definition

int main()
{
   IntegerSet a;
   IntegerSet b;
   IntegerSet c;
   IntegerSet d;

   cout << "Enter set A:\n";
   a.inputSet();
   cout << "\nEnter set B:\n";
   b.inputSet();
   /* Write call to unionOfSets for object a, passing
      b as argument and assigning the result to c */
   /* Write call to intersectionOfSets for object a,
      passing b as argument and assigning the result to d */
   cout << "\nUnion of A and B is:\n";
   c.printSet();
   cout << "Intersection of A and B is:\n";
   d.printSet();

   if ( a.isEqualTo( b ) )
      cout << "Set A is equal to set B\n";
   else
      cout << "Set A is not equal to set B\n";

   cout << "\nInserting 77 into set A...\n";
   a.insertElement( 77 );
   cout << "Set A is now:\n";
   a.printSet();

```

Fig. L 10.6 | Contents of `SetTest.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Integer Set

```
    cout << "\nDeleting 77 from set A...\n";
    a.deleteElement( 77 );
    cout << "Set A is now:\n";
    a.printSet();

    const int arraySize = 10;
    int intArray[ arraySize ] = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
    IntegerSet e( intArray, arraySize );

    cout << "\nSet e is:\n";
    e.printSet();

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

Fig. L 10.6 | Contents of `SetTest.cpp`. (Part 2 of 2.)

Problem-Solving Tips

1. Member function `intersectionOfSets` must return an `IntegerSet` object. The object that invokes this function and the argument passed to the member function should not be modified by the operation. `intersectionOfSets` should iterate over all integers an `IntegerSet` could contain (1–100) and add those integers that both `IntegerSets` contain to a temporary `IntegerSet` that will be returned.
2. Member function `deleteElement` should first verify that its argument is valid by calling utility function `validEntry`. If so, the corresponding element in the `set` array should be set to `0`; otherwise, display an error message.
3. Member function `isEqualTo` should iterate over all integers an `IntegerSet` could contain and (1–100). If any integer is found that is in one set but not the other, return `false`; otherwise return `true`.

Follow-Up Question and Activity

1. Why might it be advantageous for the `set` array to be allocated dynamically using `new []`, if the `IntegerSet` class were to be used for more general sets?

Lab Exercises

Name:

Debugging

Name: ______________________ **Date:** ______________________

Section: ______________________

The following program (Fig. L 10.7–Fig. L 10.9) does not run properly. Fix all the compilation errors so that the program compiles successfully. Once the program compiles, compare the output with that of the sample output and eliminate any logic errors that may exist. The sample output demonstrates what the program output should be once the program's code has been corrected. [*Note:* Make sure any memory allocated dynamically is deleted properly.]

Sample Output

```
There are currently 0 students

A student has been added
Here are the grades for Student 1
  100   75   89

A student has been added
Here are the grades for Student 2
   83   92

A student has been added
Here are the grades for Student 3
   62   91

There are currently 3 students

Student 2 has been deleted
Student 1 has been deleted
Student 3 has been deleted
```

Broken Code

```cpp
// Debugging:  Student.h

#ifndef STUDENT_H
#define STUDENT_H

// class Student definition
class Student
{
public:
   Student( const char * );
   ~Student();
   void displayGrades() const;
   Student addGrade( int ) const;
   static int getNumStudents();

```

Fig. L 10.7 | Contents of Student.h (Part 1 of 2).

Lab Exercises

Name:

Debugging

```
private:
   int *grades;
   char *name;
   int numGrades;
   int idNum;

   static int numStudents = 0;

}; // end class Student

#endif // STUDENT_H
```

Fig. L 10.7 | Contents of `Student.h`. (Part 2 of 2.)

```
// Debugging: Student.cpp
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;

#include "Student.h"

#include <new>
static int numStudents = 0;

// constructor
Student::Student( const char *nPtr )
{
   grades = new int[ 1 ];
   grades[ 0 ] = 0;
   name = new char[ strlen( nPtr ) + 1 ];
   strcpy( name, nPtr );
   numGrades = 0;
   ++numStudents;

   cout << "A student has been added\n";
} // end class Student constructor

// destructor
Student::~Student()
{
   cout << name << " has been deleted\n";
   delete grades;
   delete name;
   --numStudents;
} // end class Student destructor

// function displayGrades definition
void Student::displayGrades() const
{
   cout << "Here are the grades for " << name << endl;

   // output each grade
   for ( int i = 0; i < numGrades; i++ )
      cout << setw( 5 ) << grades[ i ];
```

Fig. L 10.8 | Contents of `Student.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```

   cout << endl << endl;
} // end function displayGrades

// function addGrade definition
Student Student::addGrade( int grade ) const
{
   int *temp = new int[ numGrades + 1 ];

   for ( int i = 0; i < numGrades; i++ )
      temp[ i ] = grades[ i ];

   temp[ numGrades ] = grade;
   grades = temp;
   ++numGrades;

   return this;
} // end function addGrade

// function getNumStudents definition
static int Student::getNumStudents()
{
   return numStudents;
} // end function getNumStudents
```

Fig. L 10.8 | Contents of `Student.cpp`. (Part 2 of 2.)

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

#include "Student.h"

int main()
{
   cout << "There are currently " << Student:getNumStudents()
        << " students\n\n";

   Student s1Ptr = new Student( "Student 1" );

   s1Ptr->addGrade( 100 ).addGrade( 75 ).addGrade( 89 );
   s1Ptr->displayGrades();

   Student *s2Ptr = new Student( "Student 2" );
   s2Ptr->addGrade( 83 )->addGrade( 92 );
   s2Ptr->displayGrades();

   const Student s3( "Student 3" );
   s3.addGrade( 62 )->addGrade( 91 ).displayGrades();

   cout << "There are currently " << getNumStudents()
        << " students\n\n";

   delete [] s2Ptr;
   delete s1Ptr;
} // end main
```

Fig. L 10.9 | Contents of `debugging.cpp`.

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. Consider the `Polynomial` class from the Coding Exercises in Chapter 9 of this lab manual. Which member functions in your class definition should be declared as `const`? Modify the header file so that they are `const`. The two get functions, `getCoef` and `getDegree`, should be declared as `const`.

Postlab Activities

Name:

Coding Exercises

2. After reading Section 10.6 in *C++ How to Program, Seventh Edition*, rewrite the data members of class `Polynomial` so that an arbitrarily large polynomial can be stored. [*Hint:* A `Polynomial` should be declared with a given `degree`. An array should then be allocated to accommodate degree-number of coefficients.]

3. Using the `Polynomial` class from the previous *Coding Exercise*, add a `static` variable to store the number of `Polynomial`s declared.

Postlab Activities

Name:

Coding Exercises

4. Using class `Polynomial` from *Coding Exercise 3,* write a declaration for a `friend` function to `print` a `Polynomial`.

5. Use the `new` operator to allocate dynamically an integer array of size `300`.

6. Deallocate the array allocated in *Coding Exercise 5*.

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. Create a `SavingsAccount` class. Use a `static` data member to store the `annualInterestRate` to store the annual interest rate for each of the savers. Each member of the class contains a `private` data member `savingsBalance` indicating the amount the saver currently has on deposit. Provide a member function `calculateMonthlyInterest` member function that calculates the monthly interest by multiplying the `balance` by `annualInterestRate` divided by 12; this interest should be added to `savingsBalance`. Provide a `static` member function `modifyInterestRate` that sets the `static` `annualInterestRate` to a new value. Write a driver program to test class `SavingsAccount`. Instantiate two different objects of class `SavingsAccount`, `saver1` and `saver2`, with balances of $2000.00 and $3000.00, respectively. Set `annualInterestRate` to 3 percent, then calculate the monthly interest and print the new balances for each of the savers. Then set the `annualInterestRate` to 4 percent, calculate the next month's interest and print the new balances for each of the savers.

Hints:

- The necessary data members are the account balance (represented as a `double`) and the interest rate (a `static` `double`) which applies to all `SavingsAccount` objects.
- Model two months worth of interest-accumulation, the first month at 3 percent and the second month at 4 percent.
- Sample output:

```
Initial balances:
Saver 1: $2000.00       Saver 2: $3000.00

Balances after 1 month's interest applied at .03:
Saver 1: $2005.00       Saver 2: $3007.50

Balances after 1 month's interest applied at .04:
Saver 1: $2011.68       Saver 2: $3017.53
```

2. Modify class `Date` in Fig. 10.10 to have the following capabilities:
   a) Output the date in multiple formats such as

```
DDD YYYY
MM/DD/YY
June 14, 1992
```

   b) Use overloaded constructors to create `Date` objects initialized with dates of the formats in part (a).

Postlab Activities

Name:

Programming Challenges

c) Create a `Date` constructor that reads the system date, using the standard library functions of the `<ctime>` header, and sets the `Date` members. (See your compiler's reference documentation or visit the Web site `www.cplusplus.com/ref/ctime/index.html` for information on the functions in header `<ctime>`.)

Hints:

- There are four constructors for this class: a default constructor that sets the date to the current date, using `<ctime>`; a constructor that takes a date in the form (*DDD*, *YYYY*); where *DDD* represents the day of the year, a constructor that takes a date in the form (*MM*, *DD*, *YY*) and a constructor which takes the month name, day and year. Use a `char*` and two `int`s for the last constructor.
- In addition to the four constructors, include functions for setting the `month`, `day` and `year`. No other data members are necessary.
- Write three different printing member functions. You may find it necessary to implement helper member functions that perform the following tasks:
- Return the name of a month (as a `char*`).
- Return the number of days in a month.
- Test for a leap year. A year is a leap year if it is divisible 400 or divisible by four and not by 100.
- Return the name of a month.
- Convert *DDD* to *MM DD*.
- Convert *MM DD* to *DDD*.
- Convert from month name to *MM*.
- Sample output:

```
9/13/1999
3/25/2004
9/1/2000
12/14/2004

256 1999
85 2004
245 2000
349 2004

09/13/99
03/25/04
09/01/00
12/14/04

September 13, 1999
March 25, 2004
September 1, 2000
December 14, 2004

Date object destructor for date 12/14/2004

Date object destructor for date 9/1/2000

Date object destructor for date 3/25/2004

Date object destructor for date 9/13/1999
```