Section 9.14 Using the `this` Pointer

- Every object has access to its own address through the **`this` pointer** (p. 423).
- An object's `this` pointer is not part of the object itself—i.e., the size of the memory occupied by the `this` pointer is not reflected in the result of a `sizeof` operation on the object.
- The `this` pointer is passed as an implicit argument to each non-`static` member function.
- Objects use the `this` pointer implicitly (as we've done to this point) or explicitly to reference their data members and member functions.
- The `this` pointer enables **cascaded member-function calls** (p. 425) in which multiple functions are invoked in the same statement.

Section 9.15 `static` Class Members

- A **`static` data member** (p. 429) represents "classwide" information (i.e., a property of the class shared by all instances, not a property of a specific object of the class).
- `static` data members have class scope and can be declared `public`, `private` or `protected`.
- A class's `static` members exist even when no objects of that class exist.
- To access a `public static` class member when no objects of the class exist, simply prefix the class name and the scope resolution operator (`::`) to the name of the data member.
- The `static` keyword cannot be applied to a member definition that appears outside the class definition.
- A member function should be declared **`static`** (p. 430) if it does not access non-`static` data members or non-`static` member functions of the class. Unlike non-`static` member functions, a `static` member function does not have a `this` pointer, because `static` data members and `static` member functions exist independently of any objects of a class.

Self-Review Exercises

9.1 Fill in the blanks in each of the following:

a) Class members are accessed via the ________ operator in conjunction with the name of an object (or reference to an object) of the class or via the ________ operator in conjunction with a pointer to an object of the class.

b) Class members specified as ________ are accessible only to member functions of the class and `friend`s of the class.

c) ________ class members are accessible anywhere an object of the class is in scope.

d) ________ can be used to assign an object of a class to another object of the same class.

e) A nonmember function must be declared by the class as a(n) ________ of a class to have access to that class's `private` data members.

f) A constant object must be ________; it cannot be modified after it's created.

g) A(n) ________ data member represents classwide information.

h) An object's non-`static` member functions have access to a "self pointer" to the object called the ________ pointer.

i) Keyword ________ specifies that an object or variable is not modifiable.

j) If a member initializer is not provided for a member object of a class, the object's ________ is called.

k) A member function should be `static` if it does not access ________ class members.

l) Member objects are constructed ________ their enclosing class object.

m) When a member function is defined outside the class definition, the function header must include the class name and the ________, followed by the function name to "tie" the member function to the class definition.

9.2 Find the error(s) in each of the following and explain how to correct it (them):

a) Assume the following prototype is declared in class `Time`:

```
void ~Time(int);
```

b) Assume the following prototype is declared in class `Employee`:

```
int Employee(string, string);
```

c) The following is a definition of class `Example`:

```
class Example {
public:
   Example(int y = 10) : data(y) { }

   int getIncrementedData() const {
      return ++data;
   }

   static int getCount() {
      cout << "Data is " << data << endl;
      return count;
   }
private:
   int data;
   static int count;
};
```

Answers to Self-Review Exercises

9.1 a) dot (.), arrow (->). b) `private`. c) `public`. d) Default memberwise assignment (performed by the assignment operator). e) `friend`. f) initialized. g) `static`. h) `this`. i) `const`. j) default constructor. k) non-`static`. l) before. m) `::` scope resolution operator.

9.2 a) *Error:* Destructors are not allowed to return values (or even specify a return type) or take arguments.
Correction: Remove the return type `void` and the parameter `int` from the declaration.

b) *Error:* Constructors are not allowed to return values.
Correction: Remove the return type `int` from the declaration.

c) *Error:* The class definition for `Example` has two errors. The first occurs in function `getIncrementedData`. The function is declared `const`, but it modifies the object.
Correction: To correct the first error, remove the `const` keyword from the definition of `getIncrementedData`. [*Note:* It would also be appropriate to rename this member function, as *get* functions are typically `const` member functions.]
Error: The second error occurs in function `getCount`. This function is declared `static`, so it's not allowed to access any non-`static` class member (i.e., `data`).
Correction: To correct the second error, remove the output line from the `getCount` definition.

Exercises

9.3 *(Scope Resolution Operator)* What's the purpose of the scope resolution operator?

9.4 *(Enhancing Class `Time`)* Provide a constructor that's capable of using the current time from the `time` and `localtime` functions—declared in the C++ Standard Library header `<ctime>`—to initialize an object of the `Time` class. For descriptions of C++ Standard Library headers, classes and functions, see `http://cppreference.com`.

9.5 ***(`Complex` Class)*** 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's declared. The constructor should contain default values in case no initializers are provided. Provide `public` member functions that perform the following tasks:

a) `add`—Adds two `Complex` numbers: The real parts are added together and the imaginary parts are added together.
b) `subtract`—Subtracts 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) `toString`—Returns a `string` representation of a `Complex` number in the form `(a, b)`, where `a` is the real part and `b` is the imaginary part.

In Chapter 10, you'll learn how to overload `+`, `-` and `<<` so you can write expressions like `a + b` and `a - b` and `cout << a` to add, subtract and output `Complex` objects. [*Note:* The C++ Standard Library provides its own class `complex` for complex-number arithmetic. For information on this class, visit `http://en.cppreference.com/w/cpp/numeric/complex`.]

9.6 ***(`Rational` Class)*** 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's declared. 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) `add`—Adds two `Rational` numbers. The result should be stored in reduced form.
b) `subtract`—Subtracts two `Rational` numbers. Store the result in reduced form.
c) `multiply`—Multiplies two `Rational` numbers. Store the result in reduced form.
d) `divide`—Divides two `Rational` numbers. The result should be stored in reduced form.
e) `toRationalString`—Returns a `string` representation of a `Rational` number in the form `a/b`, where `a` is the numerator and `b` is the denominator.
f) `toDouble`—Returns the `Rational` number as a `double`.

In Chapter 10, you'll learn how to overload `+`, `-`, `*`, `/` and `<<` so you can write expressions like `a + b`, `a - b`, `a * b`, `a - b` and `cout << a` to add, subtract, multiply, divide and output `Complex` objects.

9.7 ***(Enhancing Class `Time`)*** Modify the `Time` class of Figs. 9.5–9.6 to include a `tick` member function that increments the time stored in a `Time` object by one second. 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).

9.8 ***(Enhancing Class `Date`)*** Modify the `Date` class of Figs. 9.14–9.15 to perform error checking on the initializer values for data members `month`, `day` and `year`. Also, provide a member function `nextDay` to increment the day by one. Write a program that tests function `nextDay` in a loop that prints the date during each iteration to illustrate that `nextDay` works correctly. Be sure to test the following cases:

a) Incrementing into the next month.
b) Incrementing into the next year.

9.9 *(Combining Class `Time` and Class `Date`)* Combine the modified `Time` class of Exercise 9.7 and the modified `Date` class of Exercise 9.8 into one class called `DateAndTime`. Modify the `tick` function to call the `nextDay` function if the time increments into the next day. Modify functions `toStandardString` and `toUniversalString` so that each returns a `string` containing the date and time. Write a program to test the new class `DateAndTime`. Specifically, test incrementing the time into the next day.

9.10 *(Returning Error Indicators from Class `Time`'s set Functions)* Modify the *set* functions in the `Time` class of Figs. 9.5–9.6 to return appropriate error values if an attempt is made to *set* a data member of an object of class `Time` to an invalid value. Write a program that tests your new version of class `Time`. Display error messages when *set* functions return error values.

9.11 *(`Rectangle` Class)* 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.

9.12 *(Enhancing Class `Rectangle`)* Create a more sophisticated `Rectangle` class than the one you created in Exercise 9.11. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* function that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single *x*- or *y*-coordinate larger than 20.0. The *set* function also verifies that the supplied coordinates do, in fact, specify a rectangle. Provide member functions that calculate the `length`, `width`, `perimeter` and `area`. The length is the larger of the two dimensions. Include a predicate function `square` that determines whether the rectangle is a square.

9.13 *(Enhancing Class `Rectangle`)* Modify class `Rectangle` from Exercise 9.12 to include a `draw` function that displays the rectangle inside a 25-by-25 box enclosing the portion of the first quadrant in which the rectangle resides. Include a `setFillCharacter` function to specify the character out of which the body of the rectangle will be drawn. Include a `setPerimeterCharacter` function to specify the character that will be used to draw the border of the rectangle. If you feel ambitious, you might include functions to scale the size of the rectangle and move it around within the designated portion of the first quadrant.

9.14 *(`HugeInteger` Class)* Create a class `HugeInteger` that uses a 40-element array of digits to store integers as large as 40 digits each. Provide member functions `input`, `output`, `add` and `subtract`. For comparing `HugeInteger` objects, provide functions `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `isGreaterThanOrEqualTo` and `isLessThanOrEqualTo`—each of these is a "predicate" function that simply returns `true` if the relationship holds between the two `HugeInteger`s and returns `false` if the relationship does not hold. Also, provide a predicate function `isZero`. If you feel ambitious, provide member functions `multiply`, `divide` and `remainder`. In Chapter 10, you'll learn how to overload input, output, arithmetic, equality and relational operators so that you can write expressions containing `HugeInteger` objects, rather than explicitly calling member functions.

9.15 *(`TicTacToe` Class)* Create a class `TicTacToe` that will enable you to write a complete program to play the game of tic-tac-toe. The class contains as `private` data a 3-by-3 two-dimensional array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square. Place a 2 wherever the second player moves. Each move must be to an empty square. After each move, determine whether the game has been won or is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play three-dimensional tic-tac-toe on a 4-by-4-by-4 board. [*Caution:* This is an extremely challenging project that could take many weeks of effort!]

9.16 *(Friendship)* Explain the notion of friendship. Explain the negative aspects of friendship as described in the text.

9.17 *(Constructor Overloading)* Can a `Time` class definition that includes *both* of the following constructors:

```
Time(int h = 0, int m = 0, int s = 0);
Time();
```

be used to default construct a `Time` object? If not, explain why.

9.18 *(Constructors and Destructors)* What happens when a return type, even `void`, is specified for a constructor or destructor?

9.19 *(`Date` Class Modification)* Modify class `Date` in Fig. 9.18 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).

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 `http://en.cppreference.com/w/cpp/chrono/c` for information on the functions in header `<ctime>`. You might also want to check out C++11's `chrono` library at `http://en.cppreference.com/w/cpp/chrono`.

In Chapter 10, we'll be able to create operators for testing the equality of two dates and for comparing dates to determine whether one date is prior to, or after, another.

9.20 *(`SavingsAccount` Class)* Create a `SavingsAccount` class. Use a `static` data member `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 member function `calculateMonthlyInterest` that calculates the monthly interest by multiplying the `savingsBalance` 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 the `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.

9.21 *(`IntegerSet` Class)* Create class `IntegerSet` for which each object can hold integers in the range 0 through 100. Represent the set internally as a `vector` of `bool` values. Element `a[i]` is `true` if integer *i* is in the set. Element `a[j]` is `false` if integer *j* is not in the set. The default constructor initializes a set to the so-called "empty set," i.e., a set for which all elements contain `false`.

a) Provide member functions for the common set operations. For example, provide a `unionOfSets` member function that creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the result is set to `true` if that element is `true` in either or both of the existing sets, and an element of the result is set to `false` if that element is `false` in each of the existing sets).

b) 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 result is set to false if that element is false in either or both of the existing sets, and an element of the result is set to true if that element is true in each of the existing sets).
c) Provide an insertElement member function that places a new integer *k* into a set by setting a[k] to true. Provide a deleteElement member function that deletes integer *m* by setting a[m] to false.
d) Provide a toString member function that returns a set as a string containing a list of numbers separated by spaces. Include only those elements that are present in the set (i.e., their position in the vector has a value of true). Return --- for an empty set.
e) Provide an isEqualTo member function that determines whether two sets are equal.
f) Provide an additional constructor that receives an array of integers, 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.

9.22 ***(Time Class Modification)*** It would be perfectly reasonable for the Time class of Figs. 9.5–9.6 to represent the time internally as the number of seconds since midnight rather than the three integer values hour, minute and second. Clients could use the same public member functions and get the same results. Modify the Time class of Fig. 9.5 to implement the time as the number of seconds since midnight and show that there is no visible change in functionality to the clients of the class. [*Note:* This exercise nicely demonstrates the virtues of implementation hiding.]

9.23 ***(Card Shuffling and Dealing)*** Create a program to shuffle and deal a deck of cards. The program should consist of class Card, class DeckOfCards and a driver program. Class Card should provide:

a) Data members face and suit—use enumerations to represent the faces and suits.
b) A constructor that receives two enumeration constants representing the face and suit and uses them to initialize the data members.
c) Two static arrays of strings representing the faces and suits.
d) A toString function that returns the Card as a string in the form "*face* of *suit.*" You can use the + operator to concatenate strings.

Class DeckOfCards should contain:

a) An array of Cards named deck to store the Cards.
b) An integer currentCard representing the next card to deal.
c) A default constructor that initializes the Cards in the deck.
d) A shuffle function that shuffles the Cards in the deck. The shuffle algorithm should iterate through the array of Cards. For each Card, randomly select another Card in the deck and swap the two Cards.
e) A dealCard function that returns the next Card object from the deck.
f) A moreCards function that returns a bool value indicating whether there are more Cards to deal.

The driver program should create a DeckOfCards object, shuffle the cards, then deal the 52 cards—the output should be similar to Fig. 9.31.

9.24 ***(Card Shuffling and Dealing)*** Modify the program you developed in Exercise 9.23 so that it deals a five-card poker hand. Then write functions to accomplish each of the following:

a) Determine whether the hand contains a pair.
b) Determine whether the hand contains two pairs.
c) Determine whether the hand contains three of a kind (e.g., three jacks).
d) Determine whether the hand contains four of a kind (e.g., four aces).
e) Determine whether the hand contains a flush (i.e., all five cards of the same suit).

```
Six of Spades         Eight of Spades       Six of Clubs          Nine of Hearts
Queen of Hearts       Seven of Clubs        Nine of Spades        King of Hearts
Three of Diamonds     Deuce of Clubs        Ace of Hearts         Ten of Spades
Four of Spades        Ace of Clubs          Seven of Diamonds     Four of Hearts
Three of Clubs        Deuce of Hearts       Five of Spades        Jack of Diamonds
King of Clubs         Ten of Hearts         Three of Hearts       Six of Diamonds
Queen of Clubs        Eight of Diamonds     Deuce of Diamonds     Ten of Diamonds
Three of Spades       King of Diamonds      Nine of Clubs         Six of Hearts
Ace of Spades         Four of Diamonds      Seven of Hearts       Eight of Clubs
Deuce of Spades       Eight of Hearts       Five of Hearts        Queen of Spades
Jack of Hearts        Seven of Spades       Four of Clubs         Nine of Diamonds
Ace of Diamonds       Queen of Diamonds     Five of Clubs         King of Spades
Five of Diamonds      Ten of Clubs          Jack of Spades        Jack of Clubs
```

Fig. 9.31 | Sample card-shuffling-and-dealing output.

f) Determine whether the hand contains a straight (i.e., five cards of consecutive face values).

9.25 *(Project: Card Shuffling and Dealing)* Use the functions from Exercise 9.24 to write a program that deals two five-card poker hands, evaluates each hand and determines which is the better hand.

9.26 *(Project: Card Shuffling and Dealing)* Modify the program you developed in Exercise 9.25 so that it can simulate the dealer. The dealer's five-card hand is dealt "face down" so the player cannot see it. The program should then evaluate the dealer's hand, and, based on the quality of the hand, the dealer should draw one, two or three more cards to replace the corresponding number of unneeded cards in the original hand. The program should then reevaluate the dealer's hand.

9.27 *(Project: Card Shuffling and Dealing)* Modify the program you developed in Exercise 9.26 so that it handles the dealer's hand, but the player is allowed to decide which cards of the player's hand to replace. The program should then evaluate both hands and determine who wins. Now use this new program to play 20 games against the computer. Who wins more games, you or the computer? Have one of your friends play 20 games against the computer. Who wins more games? Based on the results of these games, make appropriate modifications to refine your poker-playing program. Play 20 more games. Does your modified program play a better game?

Making a Difference

9.28 *(Project: Emergency Response Class)* The North American emergency response service, *9-1-1*, connects callers to a local Public Service Answering Point (PSAP). Traditionally, the PSAP would ask the caller for identification information—including the caller's address, phone number and the nature of the emergency, then dispatch the appropriate emergency responders (such as the police, an ambulance or the fire department). *Enhanced 9-1-1* (or *E9-1-1)* uses computers and databases to determine the caller's physical address, directs the call to the nearest PSAP, and displays the caller's phone number and address to the call taker. *Wireless Enhanced 9-1-1* provides call takers with identification information for wireless calls. Rolled out in two phases, the first phase required carriers to provide the wireless phone number and the location of the cell site or base station transmitting the call. The second phase required carriers to provide the location of the caller (using technologies such as GPS). To learn more about 9-1-1, visit `http://www.fcc.gov/pshs/services/911-services/Welcome.html` and `http://people.howstuffworks.com/9-1-1.htm`.

An important part of creating a class is determining the class's attributes (data members). For this class-design exercise, research 9-1-1 services on the Internet. Then, design a class called `Emergency` that might be used in an object-oriented 9-1-1 emergency response system. List the attributes that an object of this class might use to represent the emergency. For example, the class might