```
34  int main()
35  {
36     Count counter; // create Count object
37
38     cout << "counter.x after instantiation: ";
39     counter.print();
40
41     setX( counter, 8 ); // set x using a friend function
42     cout << "counter.x after call to setX friend function: ";
43     counter.print();
44  } // end main
```

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```

Fig. 9.22 | Friends can access `private` members of a class. (Part 3 of 3.)

9.12 `friend` Functions and `friend` Classes (cont.)

- It would normally be appropriate to define function `setX` as a member function of class `Count`.
- It would also normally be appropriate to separate the program of Fig. 9.22 into three files:
  1. A header (e.g., `Count.h`) containing the `Count` class definition, which in turn contains the prototype of `friend` function `setX`
  2. An implementation file (e.g., `Count.cpp`) containing the definitions of class `Count`'s member functions and the definition of `friend` function `setX`
  3. A test program (e.g., `fig09_22.cpp`) with `main`.

9.12 `friend` Functions and `friend` Classes (cont.)

Overloaded `friend` Functions

- It's possible to specify overloaded functions as `friend`s of a class.
- Each function intended to be a `friend` must be explicitly declared in the class definition as a `friend` of the class.

Software Engineering Observation 9.12

Even though the prototypes for `friend` functions appear in the class definition, friends are not member functions.

Software Engineering Observation 9.13

Member access notions of `private`, `protected` and `public` are not relevant to `friend` declarations, so `friend` declarations can be placed anywhere in a class definition.

Good Programming Practice 9.4

Place all friendship declarations first inside the class definition's body and do not precede them with any access specifier.

9.13 Using the `this` Pointer

- Every object has access to its own address through a pointer called `this` (a C++ keyword).
- The `this` pointer is *not* part of the object itself—i.e., the memory occupied by the this pointer is not reflected in the result of a `sizeof` operation on the object.
- Rather, the this pointer is passed (by the compiler) as an *implicit* argument to each of the object's non-`static` member functions.

9.13 Using the `this` Pointer (cont.)

Using the `this` Pointer to Avoid Naming Collisions

- Member functions use the this pointer *implicitly* (as we've done so far) or *explicitly* to reference an object's data members and other member functions.
- A common *explicit* use of the `this` pointer is to avoid *naming conflicts* between a class's data members and member-function parameters (or other local variables).

9.13 Using the `this` Pointer (cont.)

- Consider the `Time` class's hour data member and `setHour` member function in Figs. 9.4–9.5.
- We could have defined `setHour` as:

```
// set hour value
void Time::setHour( int hour  )
{
   if ( hour >= 0 && hour < 24 )
      this->hour + hour; //use this pointer to access data member
   else
      throw invalid_argument( "hour must be 0-23" );
} // end function setHour
```

Error-Prevention Tip 9.4

To make your code clearer and more maintainable, and to avoid errors, never hide data members with local variable names.

9.13 Using the `this` Pointer (cont.)

Type of the `this` Pointer

- The type of the `this` pointer depends on the type of the object and whether the member function in which `this` is used is declared `const`.
- For example, in a non-`const` member function of class `Employee`, the `this` pointer has the type `Employee *`. In a `const` member function, the this pointer has the type `const Employee *`.

9.13 Using the `this` Pointer (cont.)

Implicitly and Explicitly Using the `this` Pointer to Access an Object's Data Members

- Figure 9.23 demonstrates the implicit and explicit use of the this pointer to enable a member function of class `Test` to print the private data `x` of a `Test` object.
- In the next example and in Chapter 10, we show some substantial and subtle examples of using `this`.

```
1  // Fig. 9.23: fig09_23.cpp
2  // Using the this pointer to refer to object members.
3  #include <iostream>
4  using namespace std;
5
6  class Test
7  {
8  public:
9     explicit Test( int = 0 ); // default constructor
10    void print() const;
11 private:
12    int x;
13 }; // end class Test
14
15 // constructor
16 Test::Test( int value )
17    : x( value ) // initialize x to value
18 {
19    // empty body
20 } // end constructor Test
21
```

Fig. 9.23 | using the `this` pointer to refer to object members. (Part 1 of 3.)

```
22  // print x using implicit and explicit this pointers;
23  // the parentheses around *this are required
24  void Test::print() const
25  {
26     // implicitly use the this pointer to access the member x
27     cout << "        x = " << x;
28
29     // explicitly use the this pointer and the arrow operator
30     // to access the member x
31     cout << "\n  this->x = " << this->x;
32
33     // explicitly use the dereferenced this pointer and
34     // the dot operator to access the member x
35     cout << "\n(*this).x = " << ( *this ).x << endl;
36  } // end function print
37
38  int main()
39  {
40     Test testObject( 12 ); // instantiate and initialize testObject
41
42     testObject.print();
43  } // end main
```

Fig. 9.23 | using the `this` pointer to refer to object members. (Part 2 of 3.)

```
        x = 12
  this->x = 12
(*this).x = 12
```

Fig. 9.23 | using the `this` pointer to refer to object members. (Part 3 of 3.)