9.7 When Constructors and Destructors Are Called (cont.)

Constructors and Destructors for Local Objects

- Constructors and destructors for local objects are called each time execution enters and leaves the scope of the object.
- Destructors are not called for local objects if the program terminates with a call to function `exit` or function `abort`.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.7 When Constructors and Destructors Are Called (cont.)

Constructors and Destructors for `static` Local Objects

- The constructor for a `static` local object is called only *once*, when execution first reaches the point where the object is defined—the corresponding destructor is called when `main` terminates or the program calls function `exit`.
- Global and `static` objects are destroyed in the reverse order of their creation.
- Destructors are not called for `static` objects if the program terminates with a call to function `abort`.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.7 When Constructors and Destructors Are Called (cont.)

Demonstrating When Constructors and Destructors Are Called

- The program of Figs. 9.7–9.9 demonstrates the order in which constructors and destructors are called for objects of class `CreateAndDestroy` (Fig. 9.7 and Fig. 9.8) of various storage classes in several scopes.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```cpp
// Fig. 9.7: CreateAndDestroy.h
// CreateAndDestroy class definition.
// Member functions defined in CreateAndDestroy.cpp.
#include <string>
using namespace std;

#ifndef CREATE_H
#define CREATE_H

class CreateAndDestroy
{
public:
   CreateAndDestroy( int, string ); // constructor
   ~CreateAndDestroy(); // destructor
private:
   int objectID; // ID number for object
   string message; // message describing object
}; // end class CreateAndDestroy

#endif
```

Fig. 9.7 | `CreateAndDestroy` class definition.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```cpp
// Fig. 9.8: CreateAndDestroy.cpp
// CreateAndDestroy class member-function definitions.
#include <iostream>
#include "CreateAndDestroy.h"// include CreateAndDestroy class definition
using namespace std;

// constructor sets object's ID number and descriptive message
CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
   : objectID( ID ), message( messageString )
{
   cout << "Object " << objectID << "   constructor runs   "
      << message << endl;
} // end CreateAndDestroy constructor

// destructor
CreateAndDestroy::~CreateAndDestroy()
{
   // output newline for certain objects; helps readability
   cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );

   cout << "Object " << objectID << "   destructor runs    "
      << message << endl;
} // end ~CreateAndDestroy destructor
```

Fig. 9.8 | CreateAndDestroy class member-function definitions.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```cpp
// Fig. 9.9: fig09_09.cpp
// Order in which constructors and
// destructors are called.
#include <iostream>
#include "CreateAndDestroy.h" // include CreateAndDestroy class definition
using namespace std;

void create( void ); // prototype

CreateAndDestroy first( 1, "(global before main)" ); // global object

int main()
{
    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
    CreateAndDestroy second( 2, "(local automatic in main)" );
    static CreateAndDestroy third( 3, "(local static in main)" );

    create(); // call function to create objects

    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
    CreateAndDestroy fourth( 4, "(local automatic in main)" );
    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
} // end main

```

Fig. 9.9 | Order in which constructors and destructors are called. (Part 1 of 3.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
// function to create objects
void create( void )
{
   cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
   CreateAndDestroy fifth( 5, "(local automatic in create)" );
   static CreateAndDestroy sixth( 6, "(local static in create)" );
   CreateAndDestroy seventh( 7, "(local automatic in create)" );
   cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
} // end function create
```

Fig. 9.9 | Order in which constructors and destructors are called. (Part 2 of 3.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```
Object 1   constructor runs   (global before main)

MAIN FUNCTION: EXECUTION BEGINS
Object 2   constructor runs   (local automatic in main)
Object 3   constructor runs   (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
Object 5   constructor runs   (local automatic in create)
Object 6   constructor runs   (local static in create)
Object 7   constructor runs   (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
Object 7   destructor runs    (local automatic in create)
Object 5   destructor runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
Object 4   constructor runs   (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
Object 4   destructor runs    (local automatic in main)
Object 2   destructor runs    (local automatic in main)

Object 6   destructor runs    (local static in create)
Object 3   destructor runs    (local static in main)

Object 1   destructor runs    (global before main)
```

Fig. 9.9 | Order in which constructors and destructors are called. (Part 3 of 3.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.8 `Time` Class Case Study: A Subtle Trap—Returning a Reference or a Pointer to a `private` Data Member

- A reference to an object is an alias for the name of the object and, hence, may be used on the left side of an assignment statement.
- In this context, the reference makes a perfectly acceptable *lvalue* that can receive a value.
- Unfortunately a `public` member function of a class can return a reference to a `private` data member of that class.
- Such a reference return actually makes a call to that member function an alias for the `private` data member!
  - The function call can be used in any way that the `private` data member can be used, including as an *lvalue* in an assignment statement
  - The same problem would occur if a pointer to the `private` data were to be returned by the function.
- If a function returns a reference that's declared `const`, the reference is a non-modifiable *lvalue* and cannot be used to modify the data.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

9.8 `Time` Class Case Study: A Subtle Trap—Returning a Reference or a Pointer to a `private` Data Member

- The program of Figs. 9.10–9.12 uses a simplified `Time` class (Fig. 9.10 and Fig. 9.11) to demonstrate returning a reference to a private data member with member function `badSetHour`.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```cpp
// Fig. 9.10: Time.h
// Time class declaration.
// Member functions defined in Time.cpp

// prevent multiple inclusions of header
#ifndef TIME_H
#define TIME_H

class Time
{
public:
   explicit Time( int = 0, int = 0, int = 0 );
   void setTime( int, int, int );
   unsigned int getHour() const;
   unsigned int &badSetHour( int ); // dangerous reference return
private:
   unsigned int hour;
   unsigned int minute;
   unsigned int second;
}; // end class Time

#endif
```

Fig. 9.10 | Time class declaration.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```cpp
// Fig. 9.11: Time.cpp
// Time class member-function definitions.
#include <stdexcept>
#include "Time.h" // include definition of class Time
using namespace std;

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

// set values of hour, minute and second
void Time::setTime( int h, int m, int s )
{
   // validate hour, minute and second
   if ( ( h >= 0 && h < 24 ) && ( m >= 0 && m < 60 ) &&
      ( s >= 0 && s < 60 ) )
   {
      hour = h;
      minute = m;
      second = s;
   } // end if
```

Fig. 9.11 | Time class member-function definitions. (Part 1 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```cpp
   else
      throw invalid_argument(
         "hour, minute and/or second was out of range" );
} // end function setTime

// return hour value
unsigned int Time::getHour()
{
   return hour;
} // end function getHour

// poor practice: returning a reference to a private data member.
unsigned int &Time::badSetHour( int hh )
{
   if ( hh >= 0 && hh < 24 )
      hour = hh;
   else
      throw invalid_argument( "hour must be 0-23" );

   return hour; // dangerous reference return
} // end function badSetHour
```

Fig. 9.11 | Time class member-function definitions. (Part 2 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Software Engineering Observation 9.7

Returning a reference or a pointer to a `private` data member breaks the encapsulation of the class and makes the client code dependent on the representation of the class's data. There are cases where doing this is appropriate—we'll show an example of this when we build our custom `Array` class in Section 10.10.

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

```cpp
// Fig. 9.12: fig09_12.cpp
// Demonstrating a public member function that
// returns a reference to a private data member.
#include <iostream>
#include "Time.h" // include definition of class Time
using namespace std;

int main()
{
   Time t; // create Time object

   // initialize hourRef with the reference returned by badSetHour
   int &hourRef = t.badSetHour( 20 ); // 20 is a valid hour

   cout << "Valid hour before modification: " << hourRef;
   hourRef = 30; // use hourRef to set invalid value in Time object t
   cout << "\nInvalid hour after modification: " << t.getHour();

   // Dangerous: Function call that returns
   // a reference can be used as an lvalue!
   t.badSetHour( 12 ) = 74; // assign another invalid value to hour

```

Fig. 9.12 | `public` member function that returns a reference to a `private` data member. (Part 1 of 2.)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.