Chapter 16: Classes and Data Abstraction

Outline

16.1 Introduction
16.2 Implementing a Time Abstract Data Type with a Class
16.3 Class Scope and Accessing Class Members
16.4 Separating Interface from Implementation
16.5 Controlling Access to Members
16.6 Access Functions and Utility Functions
16.7 Initializing Class Objects: Constructors
16.8 Using Default Arguments with Constructors
16.9 Using Destructors
16.10 When Constructors and Destructors Are Called
16.11 Using Data Members and Member Functions
16.12 A Subtle Trap: Returning a Reference to a private Data Member
16.13 Assignment by Default Memberwise Copy
16.14 Software Reusability

◀ ▶

Objectives

- In this chapter, you will learn:
  - To understand the software engineering concepts of encapsulation and data hiding.
  - To understand the notions of data abstraction and abstract data types (ADTs).
  - To be able to create C++ ADTs, namely classes.
  - To understand how to create, use, and destroy class objects.
  - To be able to control access to object data members and member functions.
  - To begin to appreciate the value of object orientation.

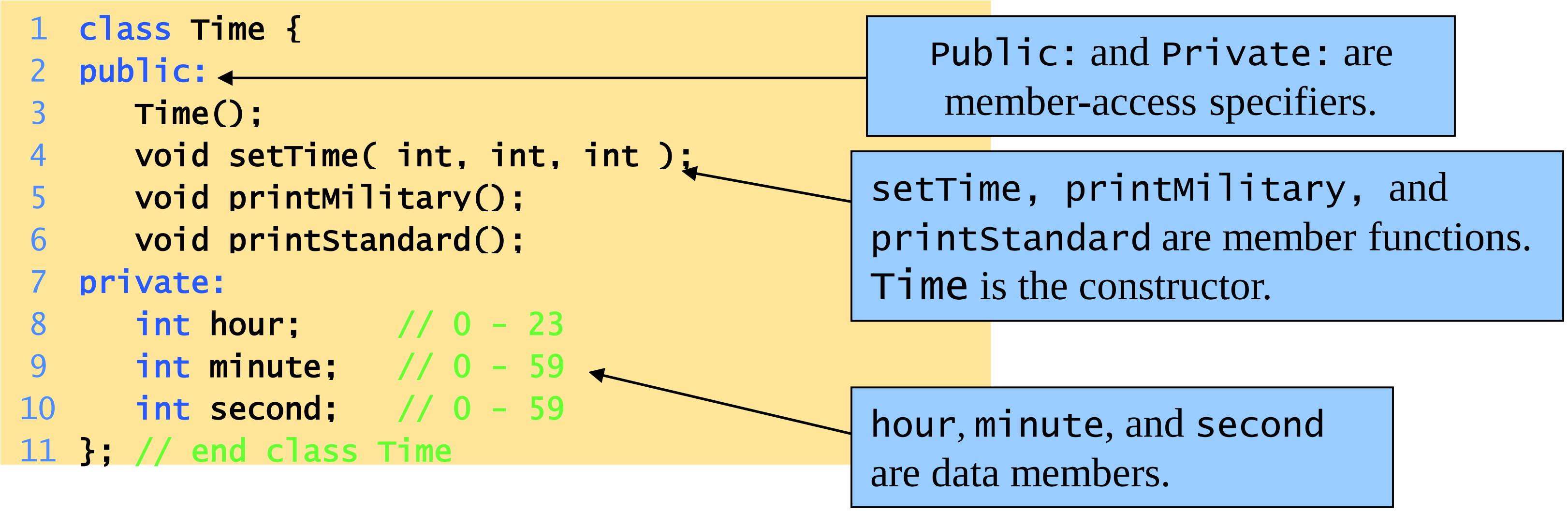
◀ ▶

16.1 Introduction

- Object-oriented programming (OOP)
  - *Encapsulates* data (attributes) and functions (behavior) into packages called *classes*
  - Data and functions closely related
- Information hiding
  - Implementation details are hidden within the classes themselves
- Unit of C++ programming: the class
  - A class is like a blueprint – reusable
  - Objects are *instantiated* (created) from the class
  - For example, a house is an instance of a “blueprint class”
  - C programmers concentrate on functions

16.2 Implementing a Time Abstract Data Type with a Class

- Classes
  - Model objects that have attributes (data members) and behaviors (member functions)
  - Defined using keyword `class`



```
class Time {
public:
   Time();
   void setTime( int, int, int );
   void printMilitary();
   void printStandard();
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
}; // end class Time
```

`Public:` and `Private:` are member-access specifiers.

`setTime`, `printMilitary`, and `printStandard` are member functions. `Time` is the constructor.

`hour`, `minute`, and `second` are data members.

16.2 Implementing a Time Abstract Data Type with a Class (II)

- Format
  - Body delineated with braces (`{` and `}`)
  - Class definition terminates with a semicolon

- Member functions and data

  `Public` - accessible wherever the program has access to an object of class `Time`

  `Private` - accessible only to member functions of the class

  `Protected` - discussed later in the course

16.2 Implementing a Time Abstract Data Type with a Class (III)

- Constructor
  - Special member function that initializes data members of a class object
  - Constructors cannot return values
  - Same name as the class
- Definitions
  - Once class defined, can be used as a data type

```
Time sunset,                   // object of type Time
     arrayOfTimes[ 5 ],        // array of Time objects
     *pointerToTime,           // pointer to a Time object
     &dinnerTime = sunset;     // reference to a Time object
```

Note: The class name becomes the new type specifier.

16.2 Implementing a Time Abstract Data Type with a Class (IV)

- Binary scope resolution operator (`::`)
  - Specifies which class owns the member function
  - Different classes can have the same name for member functions

- Format for definition class member functions

```
ReturnType ClassName::MemberFunctionName( ){
      …
      }
```

◀ ▶

16.2 Implementing a Time Abstract Data Type with a Class (V)

- If member function is defined *inside* the class
  - Scope resolution operator and class name are not needed
  - Defining a function outside a class does not change it being `public` or `private`

- Classes encourage software reuse
  - Inheritance allows new classes to be derived from old ones

- In following program
  - `Time` constructor initializes the data members to 0
    - Ensures that the object is in a consistent state when it is created

```cpp
// Fig. 16.2: fig16_02.cpp
// Time class.
#include <iostream>

using std::cout;
using std::endl;

// Time abstract data type (ADT) definition
class Time {
public:
   Time();                        // constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printMilitary();          // print military time format
   void printStandard();          // print standard time format
private:
   int hour;     // 0 - 23
   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; }

```

Outline

fig16_02.cpp (Part 1 of 3)

```cpp
// Set a new Time value using military 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

// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
} // end function printMilitary

// Print Time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard

```

fig16_02.cpp (Part 2 of 3)

fig16_02.cpp (Part 3 of 3)

```cpp
// Driver to test simple class Time
int main()
{
   Time t;  // instantiate object t of class Time

   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();

   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();

   t.setTime( 99, 99, 99 );  // attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:"
        << "\nMilitary time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
} // end function main
```

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

Program Output

16.3 Class Scope and Accessing Class Members

- Class scope
  - Data members and member functions
- File scope
  - Nonmember functions
- Function scope
  - Variables defined in member functions, destroyed after function completes
- Inside a scope
  - Members accessible by all member functions
  - Referenced by name

16.3 Class Scope and Accessing Class Members (II)

- Outside a scope
  - Use handles
    - An object name, a reference to an object or a pointer to an object
- Accessing class members
  - Same as `structs`
  - Dot (`.`) for objects and arrow (`->`) for pointers
  - Example: `t.hour` is the hour element of `t`
  - `TimePtr->hour` is the hour element

```cpp
// Fig. 16.3: fig16_03.cpp
// Demonstrating the class member access operators . and ->
//
// CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
#include <iostream>

using std::cout;
using std::endl;

// Simple class Count
class Count {
public:
   int x;
   void print() { cout << x << endl; }
}; // end class Count

int main()
{
   Count counter,                // create counter object
         *counterPtr = &counter, // pointer to counter
         &counterRef = counter;  // reference to counter

   cout << "Assign 7 to x and print using the object's name: ";
   counter.x = 7;          // assign 7 to data member x
   counter.print();        // call member function print

```

Outline

fig16_03.cpp (Part 1 of 2)

```
   cout << "Assign 8 to x and print using a reference: ";
   counterRef.x = 8;     // assign 8 to data member x
   counterRef.print();   // call member function print

   cout << "Assign 10 to x and print using a pointer: ";
   counterPtr->x = 10;   // assign 10 to data member x
   counterPtr->print(); // call member function print
   return 0;
} // end function main
```

fig16_03.cpp (Part 2 of 2)

```
Assign 7 to x and print using the object's name: 7
Assign 8 to x and print using a reference: 8
Assign 10 to x and print using a pointer: 10
```

Program Output

16.4 Separating Interface from Implementation

- Separating interface from implementation
  - Easier to modify programs
  - C++ programs can be split into

    Header files – contains class definitions and function prototypes

    Source-code files – contains member function definitions

- Program Outline:
  - Using the same `Time` class as before, create a header file
  - Create a source code file
    - Load the header file to get the class definitions
    - Define the member functions of the class

```cpp
// Fig. 16.4: time1.h
// Declaration of the Time class.
// Member functions are defined in time1.cpp

// prevent multiple inclusions of header file
#ifndef TIME1_H
#define TIME1_H

// Time abstract data type definition
class Time {
public:
   Time();                        // constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printMilitary();          // print military time format
   void printStandard();          // print standard time format
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
}; // end class Time

#endif
```

Outline

time1.h

Outline

```cpp
// Fig. 16.4: time1.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time1.h"

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

// Set a new Time value using military 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

```

time1.cpp (Part 1 of 2)

```
// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
} // end function printMilitary

// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

Outline

time1.cpp (Part 2 of 2)

```cpp
// Fig. 16.4: fig16_04.cpp
// Driver for Time1 class
// NOTE: Compile with time1.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "time1.h"

// Driver to test simple class Time
int main()
{
   Time t;  // instantiate object t of class time

   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();

   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();

```

Outline

fig16_04.cpp (Part 1 of 2)

```cpp
   t.setTime( 99, 99, 99 );  // attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:\n"
        << "Military time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
} // end function main
```

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

Outline

fig16_04.cpp (Part 2 of 2)

Program Output

16.5 Controlling Access to Members

- Purpose of `public`
  - Give clients a view of the *services* the class provides (interface)
- Purpose of `private`
  - Default setting
  - Hide details of how the class accomplishes its tasks (implementation)
  - `Private` members only accessible through the `public` interface using `public` member functions

fig16_05.cpp

```cpp
// Fig. 16.5: fig16_05.cpp
// Demonstrate errors resulting from attempts
// to access private class members.
#include <iostream>

using std::cout;

#include "time1.h"

int main()
{
   Time t;

   // Error: 'Time::hour' is not accessible
   t.hour = 7;

   // Error: 'Time::minute' is not accessible
   cout << "minute = " << t.minute;

   return 0;
} // end function main
```

Borland C++ command-line compiler error messages

```
Time1.cpp:

Fig16_05.cpp:

Error E2247 Fig16_05.cpp 15:

   'Time::hour' is not accessible in function main()

Error E2247 Fig16_05.cpp 18:

   'Time::minute' is not accessible in function main()


*** 2 errors in Compile ***
```

Microsoft Visual C++ compiler error messages

```
Compiling...
Fig16_05.cpp
D:\Fig16_05.cpp(15) : error C2248: 'hour' : cannot access private
member declared in class 'Time'
D:\Fig16_05\time1.h(18) : see declaration of 'hour'
D:\Fig16_05.cpp(18) : error C2248: 'minute' : cannot access private
member declared in class 'Time'
D:\time1.h(19) : see declaration of 'minute'
Error executing cl.exe.

test.exe - 2 error(s), 0 warning(s)
```

Program Output

16.6 Access Functions and Utility Functions

- Utility functions
  - `private` functions that support the operation of public functions
  - Not intended to be used directly by clients
- Access functions
  - `public` functions that read/display data or check conditions
  - For a container, it could call the `isEmpty` function
- Next
  - Program to take in monthly sales and output the total
  - Implementation not shown, only access functions

Outline

salesp.h

```cpp
// Fig. 16.6: salesp.h
// SalesPerson class definition
// Member functions defined in salesp.cpp
#ifndef SALESP_H
#define SALESP_H

class SalesPerson {
public:
   SalesPerson();                // constructor
   void getSalesFromUser();      // get sales figures from keyboard
   void setSales( int, double ); // User supplies one month's
                                 // sales figures.
   void printAnnualSales();

private:
   double totalAnnualSales();    // utility function
   double sales[ 12 ];           // 12 monthly sales figures
}; // end class SalesPerson

#endif
```

salesp.cpp (Part 1 of 3)

```
// Fig. 16.6: salesp.cpp
// Member functions for class SalesPerson
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::ios;

#include "salesp.h"

// Constructor function initializes array
SalesPerson::SalesPerson()
{
   for ( int i = 0; i < 12; i++ )
      sales[ i ] = 0.0;
} // end SalesPerson constructor

```

Outline

salesp.cpp (Part 2 of 3)

```cpp
// Function to get 12 sales figures from the user
// at the keyboard
void SalesPerson::getSalesFromUser()
{
   double salesFigure;

   for ( int i = 1; i <= 12; i++ ) {
      cout << "Enter sales amount for month " << i << ": ";

      cin >> salesFigure;
      setSales( i, salesFigure );
   } // end for
} // end function getSalesFromUser

// Function to set one of the 12 monthly sales figures.
// Note that the month value must be from 0 to 11.
void SalesPerson::setSales( int month, double amount )
{
   if ( month >= 1 && month <= 12 && amount > 0 )
      sales[ month - 1 ] = amount; // adjust for subscripts 0-11
   else
      cout << "Invalid month or sales figure" << endl;
} // end function setSales

```

```cpp
// Print the total annual sales
void SalesPerson::printAnnualSales()
{
   cout << setprecision( 2 )
        << setiosflags( ios::fixed | ios::showpoint )
        << "\nThe total annual sales are: $"
        << totalAnnualSales() << endl;
} // end function printAnnualSales

// Private utility function to total annual sales
double SalesPerson::totalAnnualSales()
{
   double total = 0.0;

   for ( int i = 0; i < 12; i++ )
      total += sales[ i ];

   return total;
} // end function totalAnnualSales
```

Outline

salesp.cpp (Part 3 of 3)

Outline

fig16_06.cpp

```cpp
// Fig. 16.6: fig16_06.cpp
// Demonstrating a utility function
// Compile with salesp.cpp
#include "salesp.h"

int main()
{
   SalesPerson s;         // create SalesPerson object s

   s.getSalesFromUser();  // note simple sequential code
   s.printAnnualSales();  // no control structures in main
   return 0;
} // end function main
```

Program Output

```
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.59
```

16.7 Initializing Class Objects: Constructors

- Constructor function
  - Can initialize class members
  - Same name as the class, no return type
  - Member variables can be initialized by the constructor or set afterwards

- Defining objects
  - Initializers can be provided
  - Initializers passed as arguments to the class' constructor

◀ ▶

16.7 Initializing Class Objects: Constructors (II)

- Format

  Type ObjectName(*value1, value2, …*);

  - Constructor assigns *value1*, *value2*, etc. to its member variables
  - If not enough values specified, rightmost parameters set to their default (specified by programmer)

```
myClass myObject( 3, 4.0 );
```

16.8 Using Default Arguments with Constructors

- Default constructor
  - One per class
  - Can be invoked without arguments
  - Has default arguments
- Default arguments
  - Set in default constructor function prototype (in class definition)
    - Do not set defaults in the function definition, outside of a class
  - Example:

    ```
    SampleClass( int = 0, float = 0);
    ```

    - Constructor has same name as class

```cpp
// Fig. 16.7: time2.h
// Declaration of the Time class.
// Member functions are defined in time2.cpp

// preprocessor directives that
// prevent multiple inclusions of header file
#ifndef TIME2_H
#define TIME2_H

// Time abstract data type definition
class Time {
public:
   Time( int = 0, int = 0, int = 0 );  // default constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printMilitary();          // print military time format
   void printStandard();          // print standard time format
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
}; // end class Time

#endif
```

Outline

time2.h

```cpp
// Fig. 16.7: time2.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time2.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

// Set a new Time value using military 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

```

Outline

time2.cpp (Part 1 of 2)

```
// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
} // end function printMilitary

// Print Time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

Outline

time2.cpp (Part 2 of 2)

```cpp
// Fig. 16.7: fig16_07.cpp
// Demonstrating a default constructor
// function for class Time.
#include <iostream>

using std::cout;
using std::endl;

#include "time2.h"

int main()
{
   Time t1,               // all arguments defaulted
        t2( 2 ),          // minute and second defaulted
        t3( 21, 34 ),     // second defaulted
        t4( 12, 25, 42 ), // all values specified
        t5( 27, 74, 99 ); // all bad values specified

   cout << "Constructed with:\n"
        << "all arguments defaulted:\n   ";
   t1.printMilitary();
   cout << "\n   ";
   t1.printStandard();

```

Outline

fig16_07.cpp (Part 1 of 2)

```
   cout << "\nhour specified; minute and second defaulted:"
        << "\n   ";
   t2.printMilitary();
   cout << "\n   ";
   t2.printStandard();

   cout << "\nhour and minute specified; second defaulted:"
        << "\n   ";
   t3.printMilitary();
   cout << "\n   ";
   t3.printStandard();

   cout << "\nhour, minute, and second specified:"
        << "\n   ";
   t4.printMilitary();
   cout << "\n   ";
   t4.printStandard();

   cout << "\nall invalid values specified:"
        << "\n   ";
   t5.printMilitary();
   cout << "\n   ";
   t5.printStandard();
   cout << endl;

   return 0;
} // end function main
```

Outline

fig16_07.cpp (Part 2 of 2)

```
Constructed with:
all arguments defaulted:
   00:00
   12:00:00 AM
hour specified; minute and second defaulted:
   02:00
   2:00:00 AM
hour and minute specified; second defaulted:
   21:34
   9:34:00 PM
hour, minute, and second specified:
   12:25
   12:25:42 PM
all invalid values specified:
   00:00
   12:00:00 AM
```

Program Output

16.9 Using Destructors

- Destructor
  - Member function of class
  - Performs termination housekeeping before the system reclaims the object's memory
  - Complement of the constructor
  - Name is *tilde* (~) followed by the class name
    - `~Time`
    - Recall that the constructor's name is the class name
  - Receives no parameters, returns no value
  - One destructor per class - no overloading allowed

16.10 When Constructors and Destructors Are Called

- Constructors and destructors called automatically
  - Order depends on scope of objects
- Global scope objects
  - Constructors called before any other function (including `main`)
  - Destructors called when `main` terminates (or `exit` function called)
  - Destructors not called if program terminates with `abort`

16.10 When Constructors and Destructors Are Called (II)

- Automatic local objects
  - Constructors called when objects defined
  - Destructors called when objects leave scope (when the block in which they are defined is exited)
  - Destructors not called if program ends with `exit` or `abort`
- `static` local objects
  - Constructors called when execution reaches the point where the objects are defined
  - Destructors called when `main` terminates or the `exit` function is called
  - Destructors not called if the program ends with `abort`

```
// Fig. 16.8: create.h
// Definition of class CreateAndDestroy.
// Member functions defined in create.cpp.
#ifndef CREATE_H
#define CREATE_H

class CreateAndDestroy {
public:
   CreateAndDestroy( int );  // constructor
   ~CreateAndDestroy();      // destructor
private:
   int data;
}; // end class CreateAndDestroy

#endif
```

Outline

create.h

create.cpp

```
// Fig. 16.8: create.cpp
// Member function definitions for class CreateAndDestroy
#include <iostream>

using std::cout;
using std::endl;

#include "create.h"

CreateAndDestroy::CreateAndDestroy( int value )
{
   data = value;
   cout << "Object " << data << "   constructor";
} // end CreateAndDestroy constructor

CreateAndDestroy::~CreateAndDestroy()
   { cout << "Object " << data << "   destructor " << endl; }
```

```cpp
// Fig. 16.8: fig16_08.cpp
// Demonstrating the order in which constructors and
// destructors are called.
#include <iostream>

using std::cout;
using std::endl;

#include "create.h"

void create( void );   // prototype

CreateAndDestroy first( 1 );  // global object

int main()
{
   cout << "   (global created before main)" << endl;

   CreateAndDestroy second( 2 );        // local object
   cout << "   (local automatic in main)" << endl;

   static CreateAndDestroy third( 3 );  // local object
   cout << "   (local static in main)" << endl;

   create();  // call function to create objects

```

Outline

fig16_08.cpp (Part 1 of 2)

```
   CreateAndDestroy fourth( 4 );        // local object
   cout << "   (local automatic in main)" << endl;
   return 0;
} // end function main

// Function to create objects
void create( void )
{
   CreateAndDestroy fifth( 5 );
   cout << "   (local automatic in create)" << endl;

   static CreateAndDestroy sixth( 6 );
   cout << "   (local static in create)" << endl;

   CreateAndDestroy seventh( 7 );
   cout << "   (local automatic in create)" << endl;
} // end function create
```

Outline

fig16_08.cpp (Part 2 of 2)

```
Object 1   constructor   (global created before main)
Object 2   constructor   (local automatic in main)
Object 3   constructor   (local static in main)
Object 5   constructor   (local automatic in create)
Object 6   constructor   (local static in create)
Object 7   constructor   (local automatic in create)
Object 7   destructor
Object 5   destructor
Object 4   constructor   (local automatic in main)
Object 4   destructor
Object 2   destructor
Object 6   destructor
Object 3   destructor
Object 1   destructor
```

Outline

Program Output

16.11 Using Data Members and Member Functions

- Classes provide `public` member functions
  - Set (i.e., write) or *get* (i.e., read) values of `private` data members
  - Adjustment of bank balance (a `private` data member of class `BankAccount`) by member function `computeInterest`
- Naming
  - Member function that *set*s `interestRate` typically named `setInterestRate`
  - Member function that *get*s `interestRate` would typically be called `getInterestRate`

16.11 Using Data Members and Member Functions (II)

- Do *set* and *get* capabilities effectively make data members `public`?
  - No!
  - Programmer decides what the function can set and what information the function can get
- `public` set functions should
  - Check attempts to modify data members
  - Ensure that the new value is appropriate for that data item
  - Example: an attempt to *set* the day of the month to 37 would be rejected
  - Programmer must include these features

```
// Fig. 16.9: time3.h
// Declaration of the Time class.
// Member functions defined in time3.cpp

// preprocessor directives that
// prevent multiple inclusions of header file
#ifndef TIME3_H
#define TIME3_H

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

   // set functions
   void setTime( int, int, int );  // set hour, minute, second
   void setHour( int );   // set hour
   void setMinute( int ); // set minute
   void setSecond( int ); // set second

   // get functions
   int getHour();         // return hour
   int getMinute();       // return minute
   int getSecond();       // return second

```

Outline

time3.h (Part 1 of 2)

```
   void printMilitary();  // output military time
   void printStandard();  // output standard time

private:
   int hour;              // 0 - 23
   int minute;            // 0 - 59
   int second;            // 0 - 59
}; // end class Time

#endif
```

Outline

time3.h (Part 2 of 2)

```cpp
// Fig. 16.9: time3.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time3.h"

// 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 ); }

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

```

Outline

time3.cpp (Part 1 of 3)

```
// Set the minute value
void Time::setMinute( int m )
   { minute = ( m >= 0 && m < 60 ) ? m : 0; }

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

// Get the hour value
int Time::getHour() { return hour; }

// Get the minute value
int Time::getMinute() { return minute; }

// Get the second value
int Time::getSecond() { return second; }

// Print time is military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
} // end function printMilitary

```

Outline

time3.cpp (Part 2 of 3)

```cpp
// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

Outline

time3.cpp (Part 3 of 3)

```cpp
// Fig. 16.9: fig16_09.cpp
// Demonstrating the Time class set and get functions
#include <iostream>

using std::cout;
using std::endl;

#include "time3.h"

void incrementMinutes( Time &, const int );

int main()
{
   Time t;

   t.setHour( 17 );
   t.setMinute( 34 );
   t.setSecond( 25 );

```

fig16_09.cpp (Part 1 of 3)

```cpp
   cout << "Result of setting all valid values:\n"
        << "  Hour: " << t.getHour()
        << "  Minute: " << t.getMinute()
        << "  Second: " << t.getSecond();

   t.setHour( 234 );    // invalid hour set to 0
   t.setMinute( 43 );
   t.setSecond( 6373 ); // invalid second set to 0

   cout << "\n\nResult of attempting to set invalid hour and"
        << " second:\n  Hour: " << t.getHour()
        << "  Minute: " << t.getMinute()
        << "  Second: " << t.getSecond() << "\n\n";

   t.setTime( 11, 58, 0 );
   incrementMinutes( t, 3 );

   return 0;
} // end function main

void incrementMinutes( Time &tt, const int count )
{
   cout << "Incrementing minute " << count
        << " times:\nStart time: ";
   tt.printStandard();

```

fig16_09.cpp (Part 2 of 3)

```cpp
   for ( int i = 0; i < count; i++ ) {
      tt.setMinute( ( tt.getMinute() + 1 ) % 60 );

      if ( tt.getMinute() == 0 )
         tt.setHour( ( tt.getHour() + 1 ) % 24 );

      cout << "\nminute + 1: ";
      tt.printStandard();
   } // end for

   cout << endl;
} // end function incrementMinutes
```

fig16_09.cpp (Part 3 of 3)

Program Output

```
Result of setting all valid values:
  Hour: 17  Minute: 34  Second: 25

Result of attempting to set invalid hour and second:
  Hour: 0  Minute: 43  Second: 0

Incrementing minute 3 times:
Start time: 11:58:00 AM
minute + 1: 11:59:00 AM
minute + 1: 12:00:00 PM
minute + 1: 12:01:00 PM
```

16.12 A Subtle Trap: Returning a Reference to a Private Data Member

- Reference to an object
  - Alias for the name of the object
  - May be used on the left side of an assignment statement
  - Reference can receive a value, which changes the original object as well
- One way to use this capability (unfortunately!)
  - Have a `public` member function of a class return a non-`const` reference to a `private` data member
  - This reference can be modified, which changes the original data

```cpp
// Fig. 16.10: time4.h
// Declaration of the Time class.
// Member functions defined in time4.cpp

// preprocessor directives that
// prevent multiple inclusions of header file
#ifndef TIME4_H
#define TIME4_H

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

#endif
```

Outline

time4.h

Outline

```cpp
// Fig. 16.10: time4.cpp
// Member function definitions for Time class.
#include "time4.h"

// 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 ); }

// Set the values of hour, minute, and second.
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

// Get the hour value
int Time::getHour() { return hour; }

```

time4.cpp (Part 1 of 2)

```cpp
// POOR PROGRAMMING PRACTICE:
// Returning a reference to a private data member.
int &Time::badSetHour( int hh )
{
   hour = ( hh >= 0 && hh < 24 ) ? hh : 0;

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

Outline

time4.cpp (Part 2 of 2)

fig16_10.cpp (Part 1 of 2)

```
// Fig. 16.10: fig16_10.cpp
// Demonstrating a public member function that
// returns a reference to a private data member.
// Time class has been trimmed for this example.
#include <iostream>

using std::cout;
using std::endl;

#include "time4.h"

int main()
{
   Time t;
   int &hourRef = t.badSetHour( 20 );

   cout << "Hour before modification: " << hourRef;
   hourRef = 30;  // modification with invalid value
   cout << "\nHour after modification: " << t.getHour();

```

fig16_10.cpp (Part 2 of 2)

```
   // Dangerous: Function call that returns
   // a reference can be used as an lvalue!
   t.badSetHour( 12 ) = 74;
   cout << "\n\n*********************************\n"
        << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
        << "badSetHour as an lvalue, Hour: "
        << t.getHour()
        << "\n*********************************" << endl;

   return 0;
}
```

Program Output

```
Hour before modification: 20
Hour after modification: 30

*********************************
POOR PROGRAMMING PRACTICE!!!!!!!!
badSetHour as an lvalue, Hour: 74
*********************************
```

16.13 Assignment by Default Memberwise Copy

- Assignment operator (**=**)
  - Sets variables equal, i.e., `x = y;`
  - Can be used to assign an object to another object of the same type
  - Memberwise copy — member by member copy

```
myObject1 = myObject2;
```

- Objects may be
  - Passed as function arguments
  - Returned from functions (call-by-value default)
    - Use pointers for call by reference

```cpp
// Fig. 16.11: fig16_11.cpp
// Demonstrating that class objects can be assigned
// to each other using default memberwise copy
#include <iostream>

using std::cout;
using std::endl;

// Simple Date class
class Date {
public:
   Date( int = 1, int = 1, int = 1990 ); // default constructor
   void print();
private:
   int month;
   int day;
   int year;
}; // end class Date

// Simple Date constructor with no range checking
Date::Date( int m, int d, int y )
{
   month = m;
   day = d;
   year = y;
} // end Date constructor

```

fig16_11.cpp (Part 1 of 2)

```cpp
// Print the Date in the form mm-dd-yyyy
void Date::print()
   { cout << month << '-' << day << '-' << year; }

int main()
{
   Date date1( 7, 4, 1993 ), date2;  // d2 defaults to 1/1/90

   cout << "date1 = ";
   date1.print();
   cout << "\ndate2 = ";
   date2.print();

   date2 = date1;   // assignment by default memberwise copy
   cout << "\n\nAfter default memberwise copy, date2 = ";
   date2.print();
   cout << endl;

   return 0;
} // end function main
```

```
date1 = 7-4-1993
date2 = 1-1-1990

After default memberwise copy, date2 = 7-4-1993
```

Outline

fig16_11.cpp (Part 2 of 2)

Program Output

16.14 Software Reusability

- Object-oriented programmers
  - Concentrate on implementing useful classes
- Tremendous opportunity to capture and catalog classes
  - Accessed by large segments of the programming community
  - Class libraries exist for this purpose
- Software
  - Constructed from existing, well-defined, carefully tested, portable, widely available components
  - Speeds development of powerful, high-quality software