Note: *C How to Program*, Chapter 17 is a copy of *C++ How to Program* Chapter 9. We have not renumbered the PowerPoint Slides.

Chapter 9 Classes: A Deeper Look, Part 1

C++ How to Program, 8/e

9.2 `Time` Class Case Study

- Our first example (Figs. 9.1–9.3) creates class `Time` and a driver program that tests the class.

```
// Fig. 9.1: Time.h
// Time class definition.
// Member functions are defined in Time.cpp

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

// Time class definition
class Time
{
public:
   Time(); // constructor
   void setTime( int, int, int ); // set hour, minute and second
   void printUniversal(); // print time in universal-time format
   void printStandard(); // print time in standard-time format
private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

Fig. 9.1 | Time class definition.


```
// Fig. 9.2: Time.cpp
// Member-function definitions for class Time.
#include <iostream>
#include <iomanip>
#include <stdexcept> // for invalid_argument exception class
#include "Time.h" // include definition of class Time from Time.h

using namespace std;

// Time constructor initializes each data member to zero.
Time::Time()
{
   hour = minute = second = 0;
} // end Time constructor

```

Fig. 9.2 | Time class member-function definitions. (Part 1 of 3.)

```
// set new Time value using universal time
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
   else
      throw invalid_argument(
         "hour, minute and/or second was out of range" );
} // end function setTime

```

Fig. 9.2 | Time class member-function definitions. (Part 2 of 3.)


```
// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal

// print Time in standard-time format (HH:MM:SS AM or PM)
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
      << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
      << second << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

Fig. 9.2 | Time class member-function definitions. (Part 3 of 3.)

9.2 Time Class Case Study (cont.)

- Once class `Time` has been defined, it can be used as a type in object, array, pointer and reference declarations as follows:

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



```
// Fig. 9.3: fig09_03.cpp
// Program to test class Time.
// NOTE: This file must be compiled with Time.cpp.
#include <iostream>
#include "Time.h" // include definition of class Time from Time.h
using namespace std;

int main()
{
   Time t; // instantiate object t of class Time

   // output Time object t's initial values
   cout << "The initial universal time is ";
   t.printUniversal(); // 00:00:00
   cout << "\nThe initial standard time is ";
   t.printStandard(); // 12:00:00 AM

   t.setTime( 13, 27, 6 ); // change time

```

Fig. 9.3 | Program to test class Time. (Part 1 of 3.)

```
   // output Time object t's new values
   cout << "\n\nUniversal time after setTime is ";
   t.printUniversal(); // 13:27:06
   cout << "\nStandard time after setTime is ";
   t.printStandard(); // 1:27:06 PM

   // attempt to set the time with invalid values
   try
   {
      t.setTime( 99, 99, 99 ); // all values out of range
   } // end try
   catch ( invalid_argument &e )
   {
      cout << "Exception: " << e.what() << endl << endl;
   } // end catch

   // output t's values after specifying invalid values
   cout << "\n\nAfter attempting invalid settings:"
      << "\nUniversal time: ";
   t.printUniversal(); // 00:00:00
   cout << "\nStandard time: ";
   t.printStandard(); // 12:00:00 AM
   cout << endl;
} // end main
```

Fig. 9.3 | Program to test class Time. (Part 2 of 3.)


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

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

Exception: hour, minute and/or second was out of range

After attempting invalid settings:
Universal time: 13:27:06
Standard time: 1:27:06 PM
```

Fig. 9.3 | Program to test class Time. (Part 3 of 3.)

9.6 Time Class Case Study: Constructors with Default Arguments

- Like other functions, constructors can specify default arguments.
- The default arguments to the constructor ensure that, even if no values are provided in a constructor call, the constructor still initializes the data.
- A constructor that defaults all its arguments is also a default constructor—i.e., a constructor that can be invoked with no arguments.
- There can be at most one default constructor per class.


```
// Fig. 9.8: Time.h
// Time class containing a constructor with default arguments.
// Member functions defined in Time.cpp.

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

// Time abstract data type definition
class Time
{
public:
   Time( int = 0, int = 0, int = 0 ); // default constructor

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

```

Fig. 9.8 | Time class containing a constructor with default arguments. (Part 1 of 2.)

```
// Fig. 9.9: Time.cpp
// Member-function definitions for class Time.
#include <iostream>
#include <iomanip>
#include <stdexcept>
#include "Time.h" // include definition of class Time from Time.h
using namespace std;

// Time constructor initializes each data member to zero
Time::Time( int hour, int minute, int second )
{
   setTime( hour, minute, second ); // validate and set time
} // end Time constructor

// set new Time value using universal time
void Time::setTime( int h, int m, int s )
{
   setHour( h ); // set private field hour
   setMinute( m ); // set private field minute
   setSecond( s ); // set private field second
} // end function setTime

```

Fig. 9.9 | Time class member-function definitions including a constructor that takes arguments. (Part 1 of 4.)


```
// Fig. 9.10: fig09_10.cpp
// Demonstrating a default constructor for class Time.
#include <iostream>
#include <stdexcept>
#include "Time.h" // include definition of class Time from Time.h
using namespace std;

int main()
{
   Time t1; // all arguments defaulted
   Time t2( 2 ); // hour specified; minute and second defaulted
   Time t3( 21, 34 ); // hour and minute specified; second defaulted
   Time t4( 12, 25, 42 ); // hour, minute and second specified

```

Fig. 9.10 | Constructor with default arguments. (Part 1 of 4.)

```
   cout << "Constructed with:\n\nt1: all arguments defaulted\n  ";
   t1.printUniversal(); // 00:00:00
   cout << "\n  ";
   t1.printStandard(); // 12:00:00 AM

   cout << "\n\nt2: hour specified; minute and second defaulted\n  ";
   t2.printUniversal(); // 02:00:00
   cout << "\n  ";
   t2.printStandard(); // 2:00:00 AM

   cout << "\n\nt3: hour and minute specified; second defaulted\n  ";
   t3.printUniversal(); // 21:34:00
   cout << "\n  ";
   t3.printStandard(); // 9:34:00 PM

   cout << "\n\nt4: hour, minute and second specified\n  ";
   t4.printUniversal(); // 12:25:42
   cout << "\n  ";
   t4.printStandard(); // 12:25:42 PM

```

Fig. 9.10 | Constructor with default arguments. (Part 2 of 4.)


```
   // attempt to initialize t6 with invalid values
   try
   {
      Time t5( 27, 74, 99 ); // all bad values specified
   } // end try
   catch ( invalid_argument &e )
   {
      cout << "\n\nException while initializing t5: " << e.what() << endl;
   } // end catch
} // end main
```

Fig. 9.10 | Constructor with default arguments. (Part 3 of 4.)

```
Constructed with:

t1: all arguments defaulted
  00:00:00
  12:00:00 AM

t2: hour specified; minute and second defaulted
  02:00:00
  2:00:00 AM

t3: hour and minute specified; second defaulted
  21:34:00
  9:34:00 PM

t4: hour, minute and second specified
  12:25:42
  12:25:42 PM

Exception while initializing t5: hour must be 0-23
```

Fig. 9.10 | Constructor with default arguments. (Part 4 of 4.)

9.7 Destructors

- The name of the destructor for a class is the tilde character (~) followed by the class name.
- Often referred to with the abbreviation "dtor" in the literature.
- Called implicitly when an object is destroyed.
- *The destructor itself does not actually release the object's memory—it performs termination housekeeping before the object's memory is reclaimed, so the memory may be reused to hold new objects.*
- Receives no parameters and returns no value.
- May not specify a return type—not even `void`.
- A class may have only one destructor.
- A destructor must be `public`.
- If you do not explicitly provide a destructor, the compiler creates an "empty" destructor.