Lecture 19: Classes – Part II

Class page: https://github.com/tsung-wei-huang/cs1410-40

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Review: Time Class

```
1
2   // Time class.
3   #include <iostream>
4   #include <iomanip>
5   using namespace std;
6
7   // Time class definition
8   class Time
9   {
10  public:
11     Time(); // constructor
12     void setTime( int, int, int ); // set hour, minute and second
13     void printUniversal(); // print time in universal-time format
14     void printStandard(); // print time in standard-time format
15  private:
16     int hour; // 0 - 23 (24-hour clock format)
17     int minute; // 0 - 59
18     int second; // 0 - 59
19  }; // end class Time
20
```

Review: Time Class

```
21  // Time constructor initializes each data member to zero.
22  // Ensures all Time objects start in a consistent state.
23  Time::Time()
24  {
25     hour = minute = second = 0;
26  } // end Time constructor
27
28  // set new Time value using universal time; ensure that
29  // the data remains consistent by setting invalid values to zero
30  void Time::setTime( int h, int m, int s )
31  {
32     hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
33     minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
34     second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
35  } // end function setTime
36
37  // print Time in universal-time format (HH:MM:SS)
38  void Time::printUniversal()
39  {
40     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
41        << setw( 2 ) << minute << ":" << setw( 2 ) << second;
42  } // end function printUniversal
43
```

Review: Time Class

```
44  // print Time in standard-time format (HH:MM:SS AM or PM)
45  void Time::printStandard()
46  {
47     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
48        << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
49        << second << ( hour < 12 ? " AM" : " PM" );
50  } // end function printStandard
51
52  int main()
53  {
54     Time t; // instantiate object t of class Time
55
56     // output Time object t's initial values
57     cout << "The initial universal time is ";
58     t.printUniversal(); // 00:00:00
59     cout << "\nThe initial standard time is ";
60     t.printStandard(); // 12:00:00 AM
61
62     t.setTime( 13, 27, 6 ); // change time
63
64     // output Time object t's new values
65     cout << "\n\nUniversal time after setTime is ";
66     t.printUniversal(); // 13:27:06
```

Review: Time Class

```
67      cout << "\nStandard time after setTime is ";
68      t.printStandard(); // 1:27:06 PM
69
70      t.setTime( 99, 99, 99 ); // attempt invalid settings
71
72      // output t's values after specifying invalid values
73      cout << "\n\nAfter attempting invalid settings:"
74         << "\nUniversal time: ";
75      t.printUniversal(); // 00:00:00
76      cout << "\nStandard time: ";
77      t.printStandard(); // 12:00:00 AM
78      cout << endl;
79   } // end main
```

```
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

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

Constructor with Default Arguments

- This example enhances class `Time` to demonstrate how arguments are implicitly passed to a constructor.
- Like other functions, constructors can specify default arguments.
- Line 11 declares the `Time` constructor to include default arguments (zero for each argument).
- The constructor calls member function `setTime` with the values passed to the constructor (or the default values).
- If a value is out of range, that value is set to zero (to ensure that each data member remains in a consistent state).

Constructor with Arguments

```
1
2   // Time class containing a constructor with default arguments.
3   // Member functions defined in Time.cpp.
4
5   // prevent multiple inclusions of header file
6   #ifndef TIME_H
7   #define TIME_H
8
9   // Time abstract data type definition
10  class Time
11  {
12  public:
13     Time( int = 0, int = 0, int = 0 ); // default constructor
14     void setTime( int, int, int ); // set hour, minute, second
15     void printUniversal(); // output time in universal-time format
16     void printStandard(); // output time in standard-time format
17  private:
18     int hour; // 0 - 23 (24-hour clock format)
19     int minute; // 0 - 59
20     int second; // 0 - 59
21  }; // end class Time
22
23  #endif
```

Constructors with Arguments

```
1
2  // Demonstrating a default constructor for class Time.
3  #include <iostream>
4  #include "Time.h" // include definition of class Time from Time.h
5  using namespace std;
6
7  int main()
8  {
9     Time t1; // all arguments defaulted
10    Time t2( 2 ); // hour specified; minute and second defaulted
11    Time t3( 21, 34 ); // hour and minute specified; second defaulted
12    Time t4( 12, 25, 42 ); // hour, minute and second specified
13    Time t5( 27, 74, 99 ); // all bad values specified
14
15    cout << "Constructed with:\n\nt1: all arguments defaulted\n  ";
16    t1.printUniversal(); // 00:00:00
17    cout << "\n  ";
18    t1.printStandard(); // 12:00:00 AM
19
20    cout << "\n\nt2: hour specified; minute and second defaulted\n  ";
21    t2.printUniversal(); // 02:00:00
22    cout << "\n  ";
23    t2.printStandard(); // 2:00:00 AM
```

Fig. 9.12 | Constructor with default arguments. (Part 1 of 3.)

Constructor with Default Arguments

- **Any constructor that takes no arguments is called a default constructor.**
- **A class gets a default constructor in one of two ways:**
- **The compiler implicitly creates a default constructor in a class that does not define a constructor.**
  - Such a constructor does not initialize the class's data members, but does call the default constructor for each data member that's an object of another class.
- **You explicitly define a constructor that takes no arguments.**
  - Such a default constructor will call the default constructor for each data member that's an object of another class and will perform additional initialization specified by you.
- **If you define a constructor with arguments, C++ will not implicitly create a default constructor for that class.**

Destructor

- **The name of the destructor for a class is the tilde character (~) followed by the class name.**
  - Another type of special member function.
- **The destructor is called implicitly when an object is destroyed.**
  - For example, an automatic object is destroyed when program execution leaves its scope in which that object was instantiated.
- **A destructor 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.**

When Constructor and Destructor called?

- Constructors and destructors are called implicitly by the compiler.
- The order in which these function calls occur depends on the order in which execution enters and leaves the scopes where the objects are instantiated.
- Generally, destructor calls are made in the reverse order of the corresponding constructor calls
- The storage classes of objects can alter the order in which destructors are called.

Example

```
1
2   // CreateAndDestroy class definition.
3   // Member functions defined in CreateAndDestroy.cpp.
4   #include <string>
5   using namespace std;
6
7   #ifndef CREATE_H
8   #define CREATE_H
9
10  class CreateAndDestroy
11  {
12  public:
13     CreateAndDestroy( int, string ); // constructor
14     ~CreateAndDestroy(); // destructor
15  private:
16     int objectID; // ID number for object
17     string message; // message describing object
18  }; // end class CreateAndDestroy
19
20  #endif
```

Example

```
1  // Fig. 9.14: CreateAndDestroy.cpp
2  // CreateAndDestroy class member-function definitions.
3  #include <iostream>
4  #include "CreateAndDestroy.h"// include CreateAndDestroy class definition
5  using namespace std;
6
7  // constructor
8  CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
9  {
10    objectID = ID; // set object's ID number
11    message = messageString; // set object's descriptive message
12
13    cout << "Object " << objectID << "   constructor runs    "
14       << message << endl;
15 } // end CreateAndDestroy constructor
16
17 // destructor
18 CreateAndDestroy::~CreateAndDestroy()
19 {
20    // output newline for certain objects; helps readability
21    cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
22
23    cout << "Object " << objectID << "   destructor runs    "
24       << message << endl;
25 } // end ~CreateAndDestroy destructor
```

Example

```
1
2  // Demonstrating the order in which constructors and
3  // destructors are called.
4  #include <iostream>
5  #include "CreateAndDestroy.h" // include CreateAndDestroy class definition
6  using namespace std;
7
8  void create( void ); // prototype
9
10 CreateAndDestroy first( 1, "(global before main)" ); // global object
11
12 int main()
13 {
14    cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
15    CreateAndDestroy second( 2, "(local automatic in main)" );
16    static CreateAndDestroy third( 3, "(local static in main)" );
17
18    create(); // call function to create objects
19
20    cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
21    CreateAndDestroy fourth( 4, "(local automatic in main)" );
22    cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
23 } // end main
```

Example

```
24
25  // function to create objects
26  void create( void )
27  {
28     cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;
29     CreateAndDestroy fifth( 5, "(local automatic in create)" );
30     static CreateAndDestroy sixth( 6, "(local static in create)" );
31     CreateAndDestroy seventh( 7, "(local automatic in create)" );
32     cout << "\nCREATE FUNCTION: EXECUTION ENDS" << endl;
33  } // end function create
```

Example

```
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)
```

Accessors and Modofiers

- **Classes often provide `public` member functions to allow clients of the class to *set* (i.e., assign values to) or *get* (i.e., obtain the values of) `private` data members.**
  - These member function names need not begin with `set` or `get`, but this naming convention is common.
- **This example enhances class `Time` to include public functions that allow the client code to *set* and *get* the values of the `private` data members `hour`, `minute` and `second`.**
- **The *set* functions strictly control the setting of the data members.**
- **Each *get* function simply returns the appropriate data member's value.**

Example

```
1
2   // Time class containing a constructor with default arguments.
3   // Member functions defined in Time.cpp.
4
5   // prevent multiple inclusions of header file
6   #ifndef TIME_H
7   #define TIME_H
8
9   // Time abstract data type definition
10  class Time
11  {
12  public:
13     Time( int = 0, int = 0, int = 0 ); // default constructor
14
15     // set functions
16     void setTime( int, int, int ); // set hour, minute, second
17     void setHour( int ); // set hour (after validation)
18     void setMinute( int ); // set minute (after validation)
19     void setSecond( int ); // set second (after validation)
20
```

Example

```
21      // get functions
22      int getHour(); // return hour
23      int getMinute(); // return minute
24      int getSecond(); // return second
25
26      void printUniversal(); // output time in universal-time format
27      void printStandard(); // output time in standard-time format
28   private:
29      int hour; // 0 - 23 (24-hour clock format)
30      int minute; // 0 - 59
31      int second; // 0 - 59
32   }; // end class Time
33
34   #endif
```

Example

```
1  // Fig. 9.17: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  #include <iomanip>
5  #include "Time.h" // include definition of class Time from Time.h
6  using namespace std;
7
8  // Time constructor initializes each data member to zero;
9  // ensures that Time objects start in a consistent state
10 Time::Time( int hr, int min, int sec )
11 {
12    setTime( hr, min, sec ); // validate and set time
13 } // end Time constructor
14
15 // set new Time value using universal time; ensure that
16 // the data remains consistent by setting invalid values to zero
17 void Time::setTime( int h, int m, int s )
18 {
19    setHour( h ); // set private field hour
20    setMinute( m ); // set private field minute
21    setSecond( s ); // set private field second
22 } // end function setTime
```

Example

```
23
24  // set hour value
25  void Time::setHour( int h )
26  {
27     hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
28  } // end function setHour
29
30  // set minute value
31  void Time::setMinute( int m )
32  {
33     minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
34  } // end function setMinute
35
36  // set second value
37  void Time::setSecond( int s )
38  {
39     second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
40  } // end function setSecond
41
```

Example

```
42  // return hour value
43  int Time::getHour()
44  {
45     return hour;
46  } // end function getHour
47
48  // return minute value
49  int Time::getMinute()
50  {
51     return minute;
52  } // end function getMinute
53
54  // return second value
55  int Time::getSecond()
56  {
57     return second;
58  } // end function getSecond
59
```

Example

```
60  // print Time in universal-time format (HH:MM:SS)
61  void Time::printUniversal()
62  {
63     cout << setfill( '0' ) << setw( 2 ) << getHour() << ":"
64        << setw( 2 ) << getMinute() << ":" << setw( 2 ) << getSecond();
65  } // end function printUniversal
66
67  // print Time in standard-time format (HH:MM:SS AM or PM)
68  void Time::printStandard()
69  {
70     cout << ( ( getHour() == 0 || getHour() == 12 ) ? 12 : getHour() % 12 )
71        << ":" << setfill( '0' ) << setw( 2 ) << getMinute()
72        << ":" << setw( 2 ) << getSecond() << ( hour < 12 ? " AM" : " PM" );
73  } // end function printStandard
```

Example

```
1  // Fig. 9.18: fig09_18.cpp
2  // Demonstrating the Time class set and get functions
3  #include <iostream>
4  #include "time.h"
5  using namespace std;
6
7  void incrementMinutes( Time &, const int ); // prototype
8
9  int main()
10 {
11    Time t; // create Time object
12
13    // set time using individual set functions
14    t.setHour( 17 );  // set hour to valid value
15    t.setMinute( 34 ); // set minute to valid value
16    t.setSecond( 25 ); // set second to valid value
17
18    // use get functions to obtain hour, minute and second
19    cout << "Result of setting all valid values:\n"
20         << "  Hour: " << t.getHour()
21         << "  Minute: " << t.getMinute()
22         << "  Second: " << t.getSecond();
```

Example

```
23
24      // set time using individual set functions
25      t.setHour( 234 ); // invalid hour set to 0
26      t.setMinute( 43 ); // set minute to valid value
27      t.setSecond( 6373 ); // invalid second set to 0
28
29      // display hour, minute and second after setting
30      // invalid hour and second values
31      cout << "\n\nResult of attempting to set invalid hour and"
32           << " second:\n  Hour: " << t.getHour()
33           << "  Minute: " << t.getMinute()
34           << "  Second: " << t.getSecond() << "\n\n";
35
36      t.setTime( 11, 58, 0 ); // set time
37      incrementMinutes( t, 3 ); // increment t's minute by 3
38  } // end main
39
```

Example

```
40  // add specified number of minutes to a Time object
41  void incrementMinutes( Time &tt, const int count )
42  {
43     cout << "Incrementing minute " << count
44          << " times:\nStart time: ";
45     tt.printStandard();
46
47     for ( int i = 0; i < count; i++ ) {
48        tt.setMinute( ( tt.getMinute() + 1 ) % 60 );
49
50        if ( tt.getMinute() == 0 )
51           tt.setHour( ( tt.getHour() + 1 ) % 24);
52
53        cout << "\nminute + 1: ";
54        tt.printStandard();
55     } // end for
56
57     cout << endl;
58  } // end function incrementMinutes
```

Example

```
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
```

Member Scope

- Declaring data members with access specifier `private` enforces data hiding.
- Providing `public` *set* and *get* functions allows clients of a class to access the hidden data, but only indirectly.
- The client knows that it's attempting to modify or obtain an object's data, but the client does not know how the object performs these operations.
- In some cases, a class may internally represent a piece of data one way, but expose that data to clients in a different way.
- The *set* and *get* functions allow a client to interact with an object, but the object's `private` data remains safely encapsulated (i.e., hidden) in the object itself.

Member Scope

- **`Time`'s *set* and *get* functions are called throughout the class's body.**
- **Consider changing the representation of the time from three `int` values to a single `int` value**
  - Represent the total number of seconds elapsed since midnight.
- **Only the bodies of the functions that access the `private` data directly would need to change.**
  - Those *set* and *get* functions for the `hour`, `minute` and `second`.
- **There would be no need to modify the bodies of functions `setTime`, `printUniversal` or `printStandard`.**

Member Scope

- Designing the class in this manner reduces the likelihood of programming errors when altering the class's implementation.
- Similarly, the `Time` constructor could be written to include a copy of the appropriate statements from function `setTime`.
- Doing so may be slightly more efficient, because the extra constructor call and call to `setTime` are eliminated.
- However, duplicating statements in multiple functions or constructors makes changing the class's internal data representation more difficult.
- Having the `Time` constructor call function `setTime` directly requires any changes to the implementation of `setTime` to be made only once.

Access 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.
- **One way to use this capability (unfortunately!) is to have a `public` member function of a class return a reference to a `private` data member of that class.**
- **If a function returns a `const` reference, that reference cannot be used as a modifiable *lvalue*.**

Example

```
1
2  // Time class declaration.
3  // Member functions defined in Time.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  class Time
10 {
11 public:
12    Time( int = 0, int = 0, int = 0 );
13    void setTime( int, int, int );
14    int getHour();
15    int &badSetHour( int ); // DANGEROUS reference return
16 private:
17    int hour;
18    int minute;
19    int second;
20 }; // end class Time
21
22 #endif
```

Example

```
25
26  // POOR PRACTICE: Returning a reference to a private data member.
27  int &Time::badSetHour( int hh )
28  {
29     hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
30     return hour; // DANGEROUS reference return
31  } // end function badSetHour
```

Example

```
1
2  // Demonstrating a public member function that
3  // returns a reference to a private data member.
4  #include <iostream>
5  #include "Time.h" // include definition of class Time
6  using namespace std;
7
8  int main()
9  {
10    Time t; // create Time object
11
12    // initialize hourRef with the reference returned by badSetHour
13    int &hourRef = t.badSetHour( 20 ); // 20 is a valid hour
14
15    cout << "Valid hour before modification: " << hourRef;
16    hourRef = 30; // use hourRef to set invalid value in Time object t
17    cout << "\nInvalid hour after modification: " << t.getHour();
18
19    // Dangerous: Function call that returns
20    // a reference can be used as an lvalue!
21    t.badSetHour( 12 ) = 74; // assign another invalid value to hour
22
```

Example

```
23      cout << "\n\n************************************************\n"
24         << "POOR PROGRAMMING PRACTICE!!!!!!!!\n"
25         << "t.badSetHour( 12 ) as an lvalue, invalid hour: "
26         << t.getHour()
27         << "\n************************************************" << endl;
28   } // end main
```

```
Valid hour before modification: 20
Invalid hour after modification: 30

************************************************
POOR PROGRAMMING PRACTICE!!!!!!!!
t.badSetHour( 12 ) as an lvalue, invalid hour: 74
************************************************
```

Summary

- **Constructor vs Destructor**
- **Member scope**
  - Public
  - Private
  - Protected
- **Access member data**