Lecture 11: Classes

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

Introduction

- You'll begin writing programs that employ the basic concepts of object-oriented programming.
- Typically, programs consist of function `main` and one or more classes, each containing data members and member functions.
- In this chapter, we develop a simple, well-engineered framework for organizing object-oriented programs.

Example

- **Let's begin with a simple analogy to help you understand what C++ "class" is**
- **Suppose you want to drive a car and make it go faster by pressing down on its accelerator pedal.**
- **What must happen before you can do this? Well, before you can drive a car, someone has to *design it and build it.***
- **In a sense, the pedal "hides" the complex mechanisms that actually make the car go faster.**
- **People with little or no knowledge of how cars are engineered can drive a car easily, simply by using the user-friendly "interfaces" to the car's complex internal mechanisms.**
  - Accelerator pedal, brake pedal, steering wheel, transmission shifting …

Example

- **Unfortunately, you cannot drive the engineering drawings of a car—so before you can drive a car, it must be built from the engineering drawings that describe it.**
- **A completed car will have an *actual* accelerator pedal to make the car go faster.**
- **But even that's not enough—the car will not accelerate on its own, so the driver must press the accelerator pedal to tell the car to go faster.**
- **In C++ software engineering, we use "class" to represent the abstraction**
  - Pedal => function; you don't need to know how the function is implemented but what it is offering you

Classes, Objects, and Members

- **Now let's use our car example to introduce the key object-oriented programming concepts of this section.**
- **Performing a task in a program requires a function.**
- **In C++, we create a program unit called a *class* to house a function**
  - A function belonging to a class is called a member function.
    - class => car
- **The function describes the actually performed operations.**
  - In a class, you provide one or more member functions that are designed to perform the class's tasks.
- **The function hides from its user the complex tasks.**
  - Just as the accelerator pedal of a car hides from the driver the complex mechanisms of making the car go faster.

Time Class

- We begin with an example (next page) that consists of class `Time` (lines 8–19), which represents the time of day in 24-hour clock format, the class's member functions (lines 23–50) and a `main` function (lines 52–79) that creates and manipulates a `Time` object.
- Function `main` uses this object and its member functions to set and display the time in both 24-hour and 12-hour formats.

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
```

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
```

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
```

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
```

Debrief: Time Class

- **Before creating a `Time` object, we must tell the compiler what member functions and data members belong to the class**
  - A process known as defining the class.
- **The `Time` class definition (lines 8–19) begins with keyword class followed by the class name `Time` (line 8).**
- **By convention, the name of a class begins with a capital letter**
  - Each subsequent word in the class name begins with a capital letter.
- **This capitalization style is often referred to as camel case.**
- **Every class's body is enclosed in a pair of left and right braces ({ and }), as in lines 8 and 19.**
- **The class definition terminates with a semicolon (line 19).**

Debrief: Time Class

- **Line 10 contains the access-specifier label `public:`.**
  - Access specifiers are always followed by a colon (`:`).
- **These functions appear after `public:` are the `public` member functions of the class**
  - Also known as the interface of the class.
- **We provide four public member functions in class `Time` —`Time`, `setTime`, `printUniversal` and `printStandard`.**
  - These services allow the client code to interact with an object of the class to manipulate the class's data..
- **We'll soon see that classes can have non-`public` member functions as well.**

Debrief: Time Class

- **Lines 16–18 declare three integer members to represent the hour, minute and second, respectively.**
  - These declarations appear after the access-specifier label private:.
- **Variables or functions declared after access specifier `private` (and before the next access specifier) are accessible only to member functions of the class for which they're declared.**
  - Cannot be accessed by functions outside the class (such as `main`).
- **Normally, data members are listed in the `private` portion of a class and member functions are listed in the `public` portion.**
  - Declaring data members with access specifier `private` is known as data hiding.
- **It's possible to have `private` member functions and `public` data, as we'll see later.**

Debrief: Time Class

- **The member function with the same name as the class is called a constructor.**
- **This is a special member function that initializes the data members of a class object.**
- **A class's constructor is called when a program creates an object of that class.**
- **If a class does not explicitly include a constructor, the compiler provides a default constructor**
  - A constructor with no parameters and no actions.
- **It's common to have several constructors for a class, enabling objects to be initialized several ways.**
- **Constructors cannot specify a return type.**

Debrief: Time Class

- **The `Time` constructor (lines 23–26) initializes the data members to 0—the universal-time equivalent of 12 AM.**
  - Called when the `Time` object is created to ensure that the object begins in a consistent state.
- **Invalid values cannot be stored in the data members of a `Time` object, because all subsequent attempts by a client to modify the data members are scrutinized by function `setTime`.**
- **It's strongly recommended that these data members be initialized by the class's constructor**
  - Private data members cannot be initialized directly.
- **Data members can also be assigned values by `Time`'s other member functions.**

Debrief: Time Class

- **Function `setTime` (lines 30–35) is a `public` function that declares three `int` parameters and uses them to set the time.**
- **A conditional expression tests each argument to determine whether the value is in a specified range.**
- **In class `Time`, invalid values are set to zero to ensure that the object's data values are always kept in range.**
  - Even if the provided arguments were incorrect.
- **A value passed to `setTime` is a correct value if it's in the allowed range for the member it's initializing.**
  - Ex: any number in the range 0–23 would be a correct value for the `hour`.

Debrief: Time Class

- **Function `printUniversal` (lines 38–42) takes no arguments and outputs the time in universal-time format (e.g., 13:27:06).**
- **Parameterized stream manipulator setfill specifies the fill character that appear to the left of the digits in the number.**
  - If the number being output fills the specified field, the fill character will not be displayed.
- **Once the fill character is specified with `setfill`, it applies for all subsequent values (i.e., `setfill` is a "sticky" setting).**
  - This is in contrast to `setw`, which applies only to the next value.
- **Function `printStandard` (lines 45–50) takes no arguments and outputs the date in standard-time format (e.g., 1:27:06 PM).**
- **`setfill('0')` is used to format the `minute` and `second` as two digit values with leading zeros if necessary.**

Debrief: Time Class

- **Each member-function name in the function headers (lines 23, 30, 38 and 45) is preceded by the class name and `::`, which is known as the binary scope resolution operator.**
- **This "ties" each member function to the `Time` class definition.**
  - After "tied" to the class, it is still within that class's scope.
- **Without "`Time::`" preceding each function name, these functions would not be recognized by the compiler as member functions of class `Time`**
  - The compiler would consider them as global functions, like `main`.
  - Such functions cannot access class `Time`'s `private` data or call its member functions, without specifying an object.

Debrief: Time Class

- **Typically, you cannot call a member function of a class until you create an object of that class.**
- **Line 54 creates an object of class `Time` called `t`.**
  - The variable's type is `Time`.
- **The compiler does not automatically know what type `Time` is—it's a user-defined type.**
  - We tell the compiler what `Time` is by including the class definition.
- **Each class you create becomes a new type that can be used to create objects.**
  - Can be used in object, array, pointer and reference declarations
- **When the object is instantiated (line 54), the `Time` constructor is called to initialize each `private` data member to `0`.**

Debrief: Time Class

- Lines 58 and 60 print the time in universal and standard formats to confirm the values of data members.
- These two member-function calls each use variable `t` followed by the dot operator (.), the function name and an empty set of parentheses.
- At the beginning of line 58, "`t.`" indicates that `main` should use the `Time` object that was created in line 54.
- The empty parentheses indicate that member function `printUniversal` does not require additional data to perform its task.

Debrief: Time Class

- **Note that the data members `hour`, `minute` and `second` (lines 16–18) are preceded by the `private` member access specifier.**
  - `private` data members are not accessible outside the class.
- **The philosophy here is that the data representation used within the class is of no concern to the class's clients.**
  - For example, the class can represent the time internally as the number of seconds since midnight.
  - Clients could use the same `public` member functions and get the same results without being aware of this.
- **The implementation of a class is said "*hidden from its clients*".**
- **Classes simplify programming because the user of the class object need only be concerned with the operations encapsulated in the object.**

Debrief: Time Class

- The `printUniversal` and `printStandard` member functions take no arguments, because these member functions implicitly know that they're to print the data members of the particular `Time` object for which they're invoked.
- This can make member function calls more concise than conventional function calls in procedural programming.

Summary

- **Class – fundamental to object-oriented programming**
  - Such software reuse can greatly enhance productivity and simplify code maintenance.
- **Scope**
  - Public
  - Private
- **No class next Friday 11/26 (Thanksgiving break)**
  - PA7 is due 23:59 PM 11/26
  - PA8 is due 23:50 PM 12/3
- **We will have our final lecture on 12/3**
- **Final will start on 12/10 class time – due in 24 hours**

LAB

- **Write a class that emulates a stack of integer object**
  - A stack is a data structure supporting
    - Push: insert an item into the data storage
    - Pop: remove an item from the data storage
    - Top: return (without removal) the top item from data storage
  - Use std::vector<int> as the data storage
  - You should have the following four methods
    - void Stack::Push(int item)
    - void Stack::Pop()
    - int Stack::top()
    - void Stack::print()
      - Print the present items in the stack