

# Computer Programming

## Chapter 9: Class

Hung-Yu Wei  
Department of Electrical Engineering  
National Taiwan University

# Object-Oriented Programming

- OOD (object-oriented design)
  - Think about objects
- OOP (object-oriented programming)
  - Program with objects
- Inheritance
  - Get some characteristics from existing class
- Encapsulation
  - Information hiding

# OO terminologies

- Objects (物件)
  - People, apple, car, animals
- Attributes
  - Height, weight
  - Grade in C++ class
- Behaviors/Operations
  - attend C++ class
  - Play basketball

# Class (類別)

- Class
  - People
  - student
  - NTU student
  - EE-major student
- Members of a class
  - Data member
    - Data variable in a class
  - Member function
    - Function in a class
    - Example of function:  $\sin(x)$ ,  $\cos(x)$

# Define a class

- Syntax

```
class ClassName
```

```
{
```

```
};
```

# Review: A function

- Syntax

```
Return_Type FunctionName (data input)
{
    .....
}
```

- Example

```
void displayMessage (    )
{

}
```

- Void
  - Return nothing
- Notice: this is a member function

# Data member V.S. local variable

- **Local variable**
  - Declared within a function
  - Loss its value when the function terminates
- **Data member**
  - Attribute of an object
  - Declared inside class definition
  - Exist throughout the life of an object

# Public and private

- Access specifier
  - **Public** → available for outside access
  - **Private** → cannot be accessed from outside
- Can or cannot be accessed by functions outside the object
  - Data member
  - Member function
- Why?
  - Encapsulation
  - Well-defined “interface”
  - Protect “private” information



# Your 1<sup>st</sup> Class Example: Time Class

- 3 member functions
- 3 Data members
- Access specifier
  - Public
  - Private
- Constructor

# Use class and objects

- Declaration

ClassName ObjectName;

Time t;

- Dot member selection operator •

objectName.memberFunction

objectName.dataMember

t.setTime(12, 35, 50)

# Concept: Constructor

- Constructor
  - Used to initialize an object
  - Initialize data members
- Special member function

- Same name as the class

```
Time::Time()
```

```
{
```

```
}
```

- No return type
  - Not even *void*

# Example: constructor

```
class Time
```

```
{
```

```
    public:
```

```
        Time( ) //beginning of the constructor
```

```
        {
```

```
            Initialization when a Time object is created
```

```
        } // end of the constructor
```

```
        other part of the class.....
```

```
    }
```

```
1 // Fig. 9.1: fig09_01.cpp
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
```

**Fig. 9.1** | Time class definition. (Part I of 4.)

```

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

```

```

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

```

**Fig. 9.1** | Time class definition. (Part 3 of 4.)

```

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



# Formatting --- setfill ( 'char' )

- **Setfill**

- Formatting with cout (similar to **setw**)
- Filling the gap with a character

```
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
    cout << setfill ('o') << setw (12);
    cout << 88 << endl;
    return 0;
}
```

- Review --- control statement
  - (condition) : statementTrue ? statementFalse

# Accessing Class Members (data member and member function)

- Scope resolution operator (::)
  - Indicate the member within a class scope
- Dot member selection operator (.)
  - to access the object' s data member or member function  
ObjectName.MyMemberFunction  
ObjectName.MyDataMember
- Arrow member selection operator (->)
  - preceded by a pointer to an object to access the object' s members  
ObjectPtr->MyMemberFunction  
ObjectPtr ->MyDataMember

```

1  // Fig. 9.2: fig09_02.cpp
2  // Demonstrating the class member access operators . and ->
3  #include <iostream>
4  using namespace std;
5
6  // class Count definition
7  class Count
8  {
9  public: // public data is dangerous
10     // sets the value of private data member x
11     void setX( int value )
12     {
13         x = value;
14     } // end function setX
15
16     // prints the value of private data member x
17     void print()
18     {
19         cout << x << endl;
20     } // end function print
21

```

**Fig. 9.2** | Accessing an object's member functions through each type of object handle—the object's name, a reference to the object and a pointer to the object. (Part 1 of 3.)

```

22 private:
23     int x;
24 }; // end class Count
25
26 int main()
27 {
28     Count counter; // create counter object
29     Count *counterPtr = &counter; // create pointer to counter
30     Count &counterRef = counter; // create reference to counter
31
32     cout << "Set x to 1 and print using the object's name: ";
33     counter.setX( 1 ); // set data member x to 1
34     counter.print(); // call member function print
35
36     cout << "Set x to 2 and print using a reference to an object: ";
37     counterRef.setX( 2 ); // set data member x to 2
38     counterRef.print(); // call member function print
39
40     cout << "Set x to 3 and print using a pointer to an object: ";
41     counterPtr->setX( 3 ); // set data member x to 3
42     counterPtr->print(); // call member function print
43 } // end main

```

**Fig. 9.2** | Accessing an object's member functions through each type of object handle—the object's name, a reference to the object and a pointer to the object. (Part 2 of 3.)

# Using a Class

- Use as
  - Object
  - Array
  - Pointer
  - Reference

```
int x;  
int x[5];  
int &x = y;  
int *xPtr = &z;
```

```
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
```

# Separating Interface from Implementation

- MyClass.h
  - Interface
- MyClass.cpp
  - Implementation
- Why?
  - Hiding implementation detail
  - Scalable and reusable

# Syntax: define

- Syntax – if not define

`#ifndef xxxxxx`

.....

`#endif`

- Syntax – define

`#define xxxxxx`

- Preprocessor wrapper with `#ifndef`, `#define` `#endif`
  - Goal: define and include the class definition only once
  - Usage: in header files (xxxx.h)

```
1  // Fig. 9.3: Time.h
2  // Declaration of class Time.
3  // Member functions are defined in Time.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time class definition
10 class Time
11 {
12 public:
13     Time(); // constructor
14     void setTime( int, int, int ); // set hour, minute and second
15     void printUniversal(); // print time in universal-time format
16     void printStandard(); // print 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
```

**Fig. 9.3** | Time class definition.



```

1  // Fig. 9.4: 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 all Time objects start in a consistent state.
10 Time::Time()
11 {
12     hour = minute = second = 0;
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     hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
20     minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
21     second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
22 } // end function setTime
23

```

**Fig. 9.4** | Time class member-function definitions. (Part I of 2.)

```
24 // print Time in universal-time format (HH:MM:SS)
25 void Time::printUniversal()
26 {
27     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
28         << setw( 2 ) << minute << ":" << setw( 2 ) << second;
29 } // end function printUniversal
30
31 // print Time in standard-time format (HH:MM:SS AM or PM)
32 void Time::printStandard()
33 {
34     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ) << ":"
35         << setfill( '0' ) << setw( 2 ) << minute << ":" << setw( 2 )
36         << second << ( hour < 12 ? " AM" : " PM" );
37 } // end function printStandard
```

**Fig. 9.4** | Time class member-function definitions. (Part 2 of 2.)

```

1  // Fig. 9.5: fig09_05.cpp
2  // Program to test class Time.
3  // NOTE: This file must be compiled with Time.cpp.
4  #include <iostream>
5  #include "Time.h" // include definition of class Time from Time.h
6  using namespace std;
7
8  int main()
9  {
10     Time t; // instantiate object t of class Time
11
12     // output Time object t's initial values
13     cout << "The initial universal time is ";
14     t.printUniversal(); // 00:00:00
15     cout << "\nThe initial standard time is ";
16     t.printStandard(); // 12:00:00 AM
17
18     t.setTime( 13, 27, 6 ); // change time
19
20     // output Time object t's new values
21     cout << "\n\nUniversal time after setTime is ";
22     t.printUniversal(); // 13:27:06
23     cout << "\nStandard time after setTime is ";
24     t.printStandard(); // 1:27:06 PM

```

```

25
26     t.setTime( 99, 99, 99 ); // attempt invalid settings
27
28     // output t's values after specifying invalid values
29     cout << "\n\nAfter attempting invalid settings:"
30         << "\nUniversal time: ";
31     t.printUniversal(); // 00:00:00
32     cout << "\nStandard time: ";
33     t.printStandard(); // 12:00:00 AM
34     cout << endl;
35 } // 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

**Fig. 9.5** | Program to test class Time. (Part 2 of 2.)

# 3 files

- main function (\*.cpp)
  - myClass.h
  - myClass.cpp
- 
- Compiler v.s. Linker

