

# Ngôn ngữ lập trình C++

## Chương 6 – Cấu trúc dữ liệu trừu tượng

# Chương 6: Cấu trúc dữ liệu trừu tượng

## Đề mục

- 6.1 Giới thiệu
- 6.2 Cấu trúc - struct
- 6.3 Truy nhập các thành viên của struct
- 6.4 Cài đặt kiểu dữ liệu người dùng Time bằng struct
- 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class
- 6.6 Phạm vi lớp và truy nhập các thành viên của lớp
- 6.7 Tách giao diện ra khỏi cài đặt
- 6.8 Quản lý quyền truy nhập thành viên
- 6.9 Các hàm truy nhập và các hàm tiện ích
- 6.10 Khởi tạo các đối tượng: Constructor
- 6.11 Sử dụng các đối số mặc định cho Constructor
- 6.12 Destructor - hàm hủy
- 6.13 Khi nào Constructor và Destructor được gọi
- 6.14 Sử dụng các hàm Set và Get
- 6.15 Phép gán đối tượng mặc định

## Tài liệu đọc thêm

- Day 6. TY21 (lập trình cơ bản)
- Chap 4,5. Introduction to OOP Using C++ (IOOP)  
(khái niệm hướng đối tượng)

## 6.1 Giới thiệu

- các kiểu dữ liệu phức hợp cấu tạo từ các thành phần thuộc các kiểu dữ liệu khác
  - tạo kiểu dữ liệu mới - kiểu dữ liệu người dùng tự định nghĩa (user-defined data type)
- bản ghi
  - gồm nhiều trường, mỗi trường lưu trữ một thành viên dữ liệu thuộc một kiểu dữ liệu cài sẵn hoặc một kiểu dữ liệu người dùng khác.
- ví dụ
  - Thời gian(giờ, phút, giây)      17:10:02, 04:23:12,...
  - Họ tên (họ, đệm, tên)              (Nguyễn, Văn, An), (Lê, Thị, Bình),...

## 6.1 Giới thiệu

- C++:
  - struct và class - kiểu bản ghi
  - đối tượng (một thể hiện của một kiểu struct hay class nào đó) - bản ghi
  - thành viên dữ liệu - trường
  - hàm thành viên/phương thức - thao tác trên các thành viên dữ liệu

## 6.2 Cấu trúc - struct

- **struct** definition

```
struct Time {  
    int hour;  
    int minute;  
    int second;  
};
```

Structure tag

Structure members

- quy tắc đặt tên cho các thành viên của cấu trúc
  - trong cùng **struct**: không thể trùng tên
  - trong các **struct** khác nhau: có thể trùng tên
- định nghĩa **struct** phải kết thúc bằng dấu chấm phẩy.
  - Các biến kiểu cấu trúc được khai báo như các biến thuộc các loại khác
  - Ví dụ: khai báo biến đơn, mảng, con trỏ, tham chiếu...
    - `Time timeObject;`
    - `Time timeArray[ 10 ];`
    - `Time *timePtr;`
    - `Time &timeRef = timeObject;`

## 6.2 Cấu trúc - struct

- Self-referential structure - cấu trúc đệ quy
  - thành viên của một cấu trúc không thể thuộc kiểu cấu trúc đó
  - thành viên của một cấu trúc có thể là con trỏ đến kiểu cấu trúc đó (self-referential structure - cấu trúc đệ quy)
    - sử dụng cho danh sách liên kết (linked list), hàng đợi (queue), ngăn xếp (stack), và cây (tree)

```
struct Node {  
    int data;  
    Node* next;  
};
```

## 6.3 Truy nhập các thành viên của struct

- các toán tử truy nhập thành viên (member access operator)
  - Toán tử dấu chấm ( . ) truy nhập trực tiếp đến các thành viên của cấu trúc/lớp
  - Toán tử mũi tên (->) truy nhập các thành viên qua con trỏ đến đối tượng
  - Ví dụ: in thành viên **hour** của đối tượng **timeObject**:  

```
cout << timeObject.hour;
```

hoặc  

```
timePtr = &timeObject;  
cout << timePtr->hour;
```
  - **timePtr->hour** tương đương ( **\*timePtr** ).**hour**
    - Cần có cặp ngoặc do **\*** không được ưu tiên bằng **.**



**fig06\_01.cpp**  
(1 of 3)

```
1 // Fig. 6.1: fig06_01.cpp
2 // Create a structure, set its members, and print it.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setfill;
11 using std::setw;
12
13 // structure definition
14 struct Time {
15     int hour;    // 0-23 (24-hour clock format)
16     int minute;  // 0-59
17     int second;  // 0-59
18 }; // end struct Time
19
20
21 void printUniversal( const Time & ); // prototype
22 void printStandard( const Time & ); // prototype
23
```

Định nghĩa kiểu cấu trúc **Time**  
với 3 thành viên là số nguyên.

Truyền tham chiếu tới hằng **Time**  
để tránh sao chép tham số.

Sử dụng ký hiệu dấu chấm để khởi tạo các thành viên cấu trúc.

Quyền truy nhập trực tiếp tới dữ liệu cho phép gán các giá trị không hợp lệ.

```
24 int main()
25 {
26     Time dinnerTime;           // variable of type Time
27
28     dinnerTime.hour = 18;      // set hour member of dinnerTime
29     dinnerTime.minute = 30;   // set minute member of dinnerTime
30     dinnerTime.second = 0;    // set second member of dinnerTime
31
32     cout << "Dinner will be held at ";
33     printUniversal( dinnerTime );
34     cout << " universal time,\nwhich is ";
35     printStandard( dinnerTime );
36     cout << " standard time.\n";
37
38     dinnerTime.hour = 29;      // set hour to invalid value
39     dinnerTime.minute = 73;    // set minute to invalid value
40
41     cout << "\nTime with invalid values: ";
42     printUniversal( dinnerTime );
43     cout << endl;
44
45     return 0;
46
47 } // end main
48
```

fig06\_01.cpp  
(3 of 3)

fig06\_01.cpp

Sử dụng manipulator **setfill**.

Dùng dấu chấm để truy nhập các thành viên dữ liệu.

```
49 // print time in universal-time format
50 void printUniversal( const Time &t )
51 {
52     cout << setfill( '0' ) << setw( 2 ) << t.hour << ":"
53         << setw( 2 ) << t.minute << ":"
54         << setw( 2 ) << t.second;
55 } // end function printUniversal
56
57 // print time in standard-time format
58 void printStandard( const Time &t )
59 {
60     cout << ( ( t.hour == 0 || t.hour == 12 ) ?
61             12 : t.hour % 12 ) << ":" << setfill( '0' )
62         << setw( 2 ) << t.minute << ":"
63         << setw( 2 ) << t.second
64         << ( t.hour < 12 ? " AM" : " PM" );
65 } // end function printStandard
```

Dinner will be held at 18:30:00 universal time,  
which is 6:30:00 PM standard time.

Time with invalid values: 29:73:00

## 6.4 Cài đặt kiểu dữ liệu người dùng Time bằng struct

- Truyền tham số:
  - Mặc định **struct** được truyền bằng giá trị
  - Nên truyền **struct** bằng tham chiếu để tránh được việc phải sao chép cấu trúc

## 6.4 Cài đặt kiểu dữ liệu người dùng **Time** bằng **struct**

- **struct** kiểu C
  - không có giao diện giữa bên trong và bên ngoài cấu trúc
    - Nếu cài đặt thay đổi, mọi chương trình sử dụng **struct** đó phải được sửa đổi theo
  - không thể in ra như là một biến đơn
    - Phải in/định dạng cho từng thành viên
  - không thể so sánh hai **struct** theo kiểu thông thường
    - Phải so sánh từng thành viên
- **struct** kiểu C++
  - C++ mở rộng: **struct** có chức năng như **class**
  - thông lệ: **struct** chỉ được dùng cho các cấu trúc chỉ gồm dữ liệu; **class** dùng cho các lớp có cả dữ liệu và hàm thành viên.

## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- Các lớp - Classes
  - mô hình các đối tượng
    - Thuộc tính - Attributes (data members)
    - Hành vi - Behaviors (member functions)
  - từ khoá **class**
  - các hàm thành viên – member functions
    - còn được gọi là các phương thức - method
    - được gọi để trả lời các thông điệp

Class definition bắt đầu bằng từ khoá **class**.

Class body bắt đầu bằng ngoặc mở.

Function prototype cho các **public** member function.

## Class Time definition (1 of 1)

Constructor: thành viên trùng tên với tên class, **Time**, và không có giá trị trả về.

// constructor

// set hour, minute, second

// print universal-time format

// print standard-time format

Nhãn quyền truy nhập

**private** data member chỉ có thể được truy nhập từ các member function.

Definition kết thúc bằng dấu chấm phẩy.

Class body kết thúc bằng ngoặc đóng.

```
1 class Time {
```

```
2  
3 public:
```

```
4     Time();
```

```
5     void setTime( int, int, int );
```

```
6     void printUniversal();
```

```
7     void printStandard();
```

```
8  
9 private:
```

```
10     int hour;        // 0 - 23 (24-hour clock format)
```

```
11     int minute;     // 0 - 59
```

```
12     int second;     // 0 - 59
```

```
13  
14 }; // end class Time
```

## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- Nhãn quyền truy nhập – Member access specifiers
  - quy định quyền truy nhập các thành viên của lớp từ các đoạn trình bên ngoài định nghĩa lớp
  - **public**:
    - thành viên có thể được truy nhập từ trong toàn bộ phạm vi của đối tượng
  - **private**:
    - thành viên chỉ có thể được truy nhập từ các hàm thành viên của chính lớp đó
  - **protected**:
    - dùng cho quan hệ thừa kế



## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- Constructor – phương thức khởi tạo
  - hàm thành viên đặc biệt
    - khởi tạo các thành viên dữ liệu
    - trùng tên với tên lớp
  - được gọi khi đối tượng được tạo, ví dụ khi biến được khai báo
  - có thể có vài constructor
    - hoạt động theo nguyên tắc hàm gọi chồng
  - không có giá trị trả về và không có kiểu giá trị trả về

```
class Time {  
    public:  
        Time();  
    ...  
};
```

```
...  
Time::Time()  
{  
    hour = minute = second = 0;  
} // end Time constructor
```

## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- Destructor – phương thức hủy
  - trùng tên với tên lớp
    - bắt đầu bằng dấu (~)
  - không có tham số
  - tối đa 1 destructor, không thể bị gọi chồng
  - dành cho việc dọn dẹp, chẳng hạn bộ nhớ

```
class Time {  
    public:  
        Time();  
        ~Time();  
    ...  
};
```

```
...  
Time::~~Time()  
{  
    //empty  
} // end Time destructor
```

## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- các đối tượng của một lớp
  - Kể từ sau class definition
    - tên lớp trở thành tên kiểu mới - type specifier
      - C++ là ngôn ngữ mở rộng được
    - có thể khai báo đối tượng, mảng đối tượng, con trỏ và tham chiếu tới đối tượng
  - Ví dụ:

Tên lớp trở thành tên kiểu dữ liệu mới.

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

## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- Các hàm thành viên được định nghĩa bên ngoài lớp

- toán tử phạm vi ( : : )

- gắn tên thành viên với tên lớp
  - xác định duy nhất các hàm của một lớp nào đó
  - các lớp khác nhau có thể có các hàm thành viên trùng tên

- Công thức định nghĩa hàm thành viên

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

- như nhau đối với hàm public hay private

## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- Các hàm thành viên được định nghĩa bên trong lớp
  - Không cần toán tử phạm vi (::) và tên lớp
  - Trình biên dịch sẽ chuyển thành hàm **inline** nếu có thể
    - Bên ngoài lớp, các hàm inline cần từ khoá **inline**

```
1  // Fig. 6.3: fig06_03.cpp
2  // Time class.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setfill;
11 using std::setw;
12
13 // Time abstract data type (ADT) definition
14 class Time {
15
16 public:
17     Time() ;                // constructor
18     void setTime( int, int, int ); // set hour, minute, second
19     void printUniversal() ;    // print universal-time format
20     void printStandard() ;    // print standard-time format
21
```

Định nghĩa lớp **Time**.



fig06\_03.cpp  
(2 of 5)

```
22 private:
23     int hour;        // 0 - 23 (24-hour clock format)
24     int minute;      // 0 - 59
25     int second;      // 0 - 59
26
27 }; // end class Time
28
29 // Time constructor initializes each data member to zero and
30 // ensures all Time objects start in a consistent state
31 Time::Time()
32 {
33     hour = minute = second = 0;
34
35 } // end Time constructor
36
37 // set new Time value using universal time, perform validity
38 // checks on the data values and set invalid values to zero
39 void Time::setTime( int h, int m, int s )
40 {
41     hour = ( h >= 0 && h < 24 ) ? h : 0;
42     minute = ( m >= 0 && m < 60 ) ? m : 0;
43     second = ( s >= 0 && s < 60 ) ? s : 0;
44
45 } // end function setTime
46
```

Constructor khởi tạo các thành viên dữ liệu **private** về 0.

Hàm thành viên **public** kiểm tra tính hợp lệ của giá trị các đối số trước khi gán trị cho các thành viên dữ liệu **private**

fig06\_03.cpp  
(3 of 5)

```
47 // print Time in universal format
48 void Time::printUniversal()
49 {
50     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
51         << setw( 2 ) << minute << ":"
52         << setw( 2 ) << second;
53
54 } // end function printUniversal
55
56 // print Time in standard format
57 void Time::printStandard()
58 {
59     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
60         << ":" << setfill( '0' ) << setw( 2 ) << minute
61         << ":" << setw( 2 ) << second
62         << ( hour < 12 ? " AM" : " PM" );
63
64 } // end function printStandard
65
66 int main()
67 {
68     Time t; // instantiate object t of class Time
69
```

Không có tham số (ngầm hiểu mục đích là in các thành viên dữ liệu); lời gọi hàm thành viên ngắn gọn hơn lời gọi hàm thường.

Khai báo biến `t` là đối tượng thuộc lớp `Time`.



fig06\_03.cpp

```
70 // output Time object t's initial values
71 cout << "The initial universal time is ";
72 t.printUniversal(); // 00:00:00
73
74 cout << "\n\nThe initial standard time is ";
75 t.printStandard(); // 12:00:00 AM
76
77 t.setTime( 13, 27, 6 ); // change time
78
79 // output Time object t's new values
80 cout << "\n\nUniversal time after setTime is ";
81 t.printUniversal(); // 13:27:06
82
83 cout << "\n\nStandard time after setTime is ";
84 t.printStandard(); // 1:27:06 PM
85
86 t.setTime( 99, 99, 99 ); // attempt invalid settings
87
88 // output t's values after specifying invalid values
89 cout << "\n\nAfter attempting invalid settings:"
90 << "\n\nUniversal time: ";
91 t.printUniversal(); // 00:00:00
92
```

Gọi các hàm thành viên **public** để in thời gian.

Dùng hàm thành viên **public** để gán trị cho các thành viên dữ liệu.

Thử gán các giá trị không hợp lệ cho các thành viên dữ liệu bằng cách sử dụng hàm thành viên **public**

```

93     cout << "\nStandard time: ";
94     t.printStandard();      // 12:00:00 AM
95     cout << endl;
96
97     return 0;
98
99 } // end main

```

**fig06\_03.cpp**  
(5 of 5)

**fig06\_03.cpp**  
output (1 of 1)

```

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

```

Các thành viên dữ liệu được  
gán về 0 sau khi thử các giá  
trị không hợp lệ.

## 6.5 Cài đặt một kiểu dữ liệu trừu tượng Time bằng một lớp - class

- lợi ích khi dùng lớp
  - đơn giản hóa việc lập trình
  - các giao diện – Interfaces
    - che dấu phần cài đặt – Hide implementation
  - tái sử dụng phần mềm – Software reuse
    - khả năng tích hợp – Composition (aggregation)
      - các thành viên của một lớp có thể là đối tượng thuộc lớp khác
    - thừa kế - Inheritance
      - các lớp mới được tạo từ lớp cũ

## 6.6 Phạm vi lớp và truy nhập các thành viên của lớp

- phạm vi lớp – Class scope
  - gồm thành viên dữ liệu và hàm thành viên của lớp
  - bên trong phạm vi lớp
    - Các thành viên của lớp
      - có thể được truy nhập thẳng từ mọi hàm thành viên
      - gọi bằng tên
  - bên ngoài phạm vi lớp
    - được gọi đến bằng tên đối tượng, tham chiếu/con trỏ tới đối tượng
      - `objectTime.printStandard()`

## 6.6 Phạm vi lớp và truy nhập các thành viên của lớp

- Phạm vi file - File scope
  - áp dụng cho các hàm không phải thành viên
- Phạm vi hàm – Function scope
  - Gồm các biến được khai báo trong hàm thành viên
  - chỉ được biết đến trong hàm đó
  - bị hủy khi hàm kết thúc
  - các biến trùng tên với biến thuộc phạm vi lớp
    - biến thuộc phạm vi lớp (class-scope variable) bị che (“hidden”)
      - truy nhập bằng toán tử phạm vi (::)

***ClassName :: classVariableName***

## 6.6 Phạm vi lớp và truy nhập các thành viên của lớp

- Các toán tử để truy nhập các thành viên của đối tượng
  - giống các toán tử dành cho **struct**
  - toán tử (.) dùng cho
    - đối tượng
    - tham chiếu đến đối tượng
  - toán tử (->) dùng cho
    - các con trỏ tới đối tượng

fig06\_04.cpp  
(1 of 2)

```
1 // Fig. 6.4: fig06_04.cpp
2 // Demonstrating the class member access operators . and ->
3 //
4 // CAUTION: IN FUTURE EXAMPLES WE AVOID PUBLIC DATA!
5 #include <iostream>
6
7 using std::cout;
8 using std::endl;
9
10 // class Count definition
11 class Count {
12
13 public:
14     int x;
15
16     void print()
17     {
18         cout << x << endl;
19     }
20
21 }; // end class Count
22
```

Thành viên dữ liệu **public x**  
minh họa các toán tử truy nhập;  
thông thường các thành viên dữ liệu  
đều là **private**.

```

23 int main()
24 {
25     Count counter;           // create counter object
26     Count *counterPtr = &counter;
27     Count &counterRef = counter;
28
29     cout << "Assign 1 to x and print using the object's name: ";
30     counter.x = 1;           // assign 1 to data member x
31     counter.print();         // call member function print
32
33     cout << "Assign 2 to x and print using a reference: ";
34     counterRef.x = 2;        // assign 2 to data member x
35     counterRef.print();      // call member function print
36
37     cout << "Assign 3 to x and print using a pointer: ";
38     counterPtr->x = 3;        // assign 3 to data member x
39     counterPtr->print();      // call member function print
40
41     return 0;
42 } // end main
43

```

Sử dụng dấu chấm cho đối tượng **counter**.

fig06\_04.cpp  
(2 of 2)

Sử dụng dấu chấm cho **counterRef** là tham chiếu đến đối tượng.

fig06\_04.cpp  
output (1 of 1)

Sử dụng mũi tên cho **counterPtr** là con trỏ tới đối tượng.

```

Assign 1 to x and print using the object's name: 1
Assign 2 to x and print using a reference: 2
Assign 3 to x and print using a pointer: 3

```



## 6.7 Tách giao diện ra khỏi cài đặt

- Tách giao diện khỏi cài đặt
  - ích lợi
    - dễ sửa đổi chương trình
  - bất lợi
    - phải tạo các file header gồm
      - một phần của cài đặt
        - Inline member functions – các hàm inline
      - gợi ý về phần khác của cài đặt
        - private members

## 6.7 Tách giao diện ra khỏi cài đặt

- Các file header
  - chứa các định nghĩa lớp và các nguyên mẫu hàm
  - được include trong mỗi file sử dụng lớp đó
    - **#include**
  - mở rộng của file **.h**
- Các file mã nguồn – Source-code files
  - chứa định nghĩa của các hàm thành viên
  - trùng tên file với file header tương ứng (không kể phần mở rộng)
    - đây chỉ là thông lệ, không bắt buộc
  - được biên dịch và liên kết với file chương trình chính

```

1 // Fig. 6.5: time1.h
2 // Declaration of class Time.
3 // Member functions are defined in time1.cpp
4
5 // prevent multiple inclusions of header file
6 #ifndef TIME1_H
7 #define TIME1_H
8
9 // Time abstract data type definition
10 class Time {
11
12 public:
13     Time(); // constructor
14     void setTime( int, int, int ); // set hour, minute, second
15     void printUniversal(); // print universal-time format
16     void printStandard(); // print standard-time format
17
18 private:
19     int hour; // 0 - 23 (24-hour clock format)
20     int minute; // 0 - 59
21     int second; // 0 - 59
22
23 }; // end class Time
24
25 #endif

```

Mã tiền xử lý để tránh việc file bị include nhiều lần.

"If not defined"

Mã giữa hai định hướng này không được include nên tên **TIME1\_H** đã được định nghĩa.

Định hướng tiền xử lý định nghĩa tên **TIME1\_H**.

Thông lệ đặt tên: tên header file với dấu gạch dưới thay cho dấu chấm.

## time1.cpp (1 of 3)

```
1 // Fig. 6.6: time1.cpp
2 // Member-function definitions for class Time.
3 #include <iostream>
4
5 using std::cout;
6
7 #include <iomanip>
8
9 using std::setfill;
10 using std::setw;
11
12 // include definition of class Time from time1.h
13 #include "time1.h"
14
15 // Time constructor initializes each data member to zero.
16 // Ensures all Time objects start in a consistent state.
17 Time::Time()
18 {
19     hour = minute = second = 0;
20 } // end Time constructor
21
22
```

Include header file  
**time1.h.**

Tên của header file đặt trong ngoặc kép;  
cặp ngoặc nhọn làm trình biên dịch cho  
rằng đó là một phần của thư viện chuẩn  
C++ (C++ Standard Library).

## time1.cpp (2 of 3)

```
23 // Set new Time value using universal time. Perform validity
24 // checks on the data values. Set invalid values to zero.
25 void Time::setTime( int h, int m, int s )
26 {
27     hour = ( h >= 0 && h < 24 ) ? h : 0;
28     minute = ( m >= 0 && m < 60 ) ? m : 0;
29     second = ( s >= 0 && s < 60 ) ? s : 0;
30
31 } // end function setTime
32
33 // print Time in universal format
34 void Time::printUniversal()
35 {
36     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
37         << setw( 2 ) << minute << ":"
38         << setw( 2 ) << second;
39
40 } // end function printUniversal
41
```

```
42 // print Time in standard format
43 void Time::printStandard()
44 {
45     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
46         << ":" << setfill( '0' ) << setw( 2 ) << minute
47         << ":" << setw( 2 ) << second
48         << ( hour < 12 ? " AM" : " PM" );
49
50 } // end function printStandard
```

**fig06\_07.cpp**  
(1 of 2)

```
1 // Fig. 6.7: fig06_07.cpp
2 // Program to test class Time.
3 // NOTE: This file must be compiled with time1.cpp.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 // include definition of class Time from time1.h
10 #include "time1.h"
11
12 int main()
13 {
14     Time t; // instantiate object t of class Time
15
16     // output Time object t's initial values
17     cout << "The initial universal time is ";
18     t.printUniversal(); // 00:00:00
19     cout << "\nThe initial standard time is ";
20     t.printStandard(); // 12:00:00 AM
21
22     t.setTime( 13, 27, 6 ); // change time
23
```

Include **time1.h** để đảm bảo tạo đúng và để tính kích thước đối tượng thuộc lớp **Time**.

```

24 // output Time object t's new values
25 cout << "\n\nUniversal time after setTime is ";
26 t.printUniversal(); // 13:27:06
27 cout << "\n\nStandard time after setTime is ";
28 t.printStandard(); // 1:27:06 PM
29
30 t.setTime( 99, 99, 99 ); // attempt invalid settings
31
32 // output t's values after specifying invalid values
33 cout << "\n\nAfter attempting invalid settings:"
34 << "\n\nUniversal time: ";
35 t.printUniversal(); // 00:00:00
36 cout << "\n\nStandard time: ";
37 t.printStandard(); // 12:00:00 AM
38 cout << endl;
39
40 return 0;
41
42 } // end main

```

**fig06\_07.cpp**  
(2 of 2)

**fig06\_07.cpp**  
output (1 of 1)

```

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

```



## 6.8 Quản lý quyền truy nhập thành viên

- các kiểu truy nhập – Access
  - **private**
    - kiểu mặc định - Default access mode
    - chỉ có các hàm thành viên và các hàm **friend** là có thể truy nhập các thành viên **private**
  - **public**
    - truy nhập được từ mọi hàm trong chương trình.
  - **protected**
    - dành cho quan hệ thừa kế, hiện tại chưa nói đến

fig06\_08.cpp  
(1 of 1)

```
1 // Fig. 6.8: fig06_08.cpp
2 // Demonstrate errors resulting from attempts
3 // to access private class members.
4 #include <iostream>
5
6 using std::cout;
7
8 // include definition of class Time from time1.h
9 #include "time1.h"
10
11 int main()
12 {
13     Time t; // create Time object
14
15     t.hour = 7; // error: 'Time::hour' is not accessible
16
17     // error: 'Time::minute' is not accessible
18     cout << "minute = " << t.minute;
19
20     return 0;
21
22 } // end main
23
```

**hour** là thành viên **private**;  
truy nhập các thành viên **private** sẽ gây lỗi.

**minute** cũng là **private**;

## 6.8 Quản lý quyền truy nhập thành viên

- quyền truy nhập các thành viên của **class**
  - mặc định **private**
  - phải đặt tường minh **public**, **protected**
- quyền truy nhập các thành viên của **struct**
  - mặc định **public**
  - phải đặt tường minh **private**, **protected**
- truy nhập dữ liệu **private** của lớp
  - các hàm truy nhập (accessor method)
    - Get function – hàm đọc dữ liệu
      - đọc dữ liệu **private**
    - Set function – hàm ghi dữ liệu
      - ghi dữ liệu **private**

## 6.9 Các hàm truy nhập và các hàm tiện ích

- Các hàm truy nhập – Access functions
  - **public**
  - các hàm đọc và hiển thị dữ liệu
  - các hàm ghi dữ liệu (kèm kiểm tra tính hợp lệ)
  - các hàm mệnh đề – Predicate functions
    - kiểm tra các điều kiện
- Các hàm tiện ích – Utility functions
  - **private**
  - chỉ hỗ trợ hoạt động của các hàm thành viên kiểu **public**
  - không nhằm mục đích để cho client trực tiếp sử dụng

```
1 // Fig. 6.9: salesp.h
2 // SalesPerson class definition.
3 // Member functions defined in salesp.cpp.
4 #ifndef SALESP_H
5 #define SALESP_H
6
7 class SalesPerson {
8
9 public:
10     SalesPerson(); // constructor
11     void getSalesFromUser(); // input sales from keyboard
12     void setSales( int, double ); // set sales for a month
13     void printAnnualSales(); // summarize and print sales
14
15 private:
16     double totalAnnualSales(); // utility function
17     double sales[ 12 ]; // 12 monthly sales figures
18
19 }; // end class SalesPerson
20
21 #endif
```

hàm ghi dữ liệu thực hiện việc kiểm tra tính hợp lệ của dữ liệu (validity checks).

hàm tiện ích **private**

## salesp.cpp (1 of 3)

```
1  // Fig. 6.10: salesp.cpp
2  // Member functions for class SalesPerson.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::fixed;
9
10 #include <iomanip>
11
12 using std::setprecision;
13
14 // include SalesPerson class definition from salesp.h
15 #include "salesp.h"
16
17 // initialize elements of array sales to 0.0
18 SalesPerson::SalesPerson()
19 {
20     for ( int i = 0; i < 12; i++ )
21         sales[ i ] = 0.0;
22
23 } // end SalesPerson constructor
24
```

```

25 // get 12 sales figures from the user at the keyboard
26 void SalesPerson::getSalesFromUser()
27 {
28     double salesFigure;
29
30     for ( int i = 1; i <= 12; i++ ) {
31         cout << "Enter sales amount for month " << i << ": ";
32         cin >> salesFigure;
33         setSales( i, salesFigure );
34
35     } // end for
36
37 } // end function getSalesFromUser
38
39 // set one of the 12 monthly sales figures; function subtracts
40 // one from month value for proper subscript in sales array
41 void SalesPerson::setSales( int month, double amount )
42 {
43     // test for valid month and amount values
44     if ( month >= 1 && month <= 12 && amount > 0 )
45         sales[ month - 1 ] = amount; // adjust for subscripts 0-11
46
47     else // invalid month or amount value
48         cout << "Invalid month or sales figure" << endl;

```

hàm ghi dữ liệu thực hiện việc kiểm tra tính hợp lệ của dữ liệu (validity checks).

```

49
50 } // end function setSales
51
52 // print total annual sales (with help of utility function)
53 void SalesPerson::printAnnualSales()
54 {
55     cout << setprecision( 2 ) << fixed
56         << "\nThe total annual sales are: $"
57         << totalAnnualSales() << endl; // call utility function
58
59 } // end function printAnnualSales
60
61 // private utility function to total annual sales
62 double SalesPerson::totalAnnualSales()
63 {
64     double total = 0.0; // initialize total
65
66     for ( int i = 0; i < 12; i++ ) // summarize sales results
67         total += sales[ i ];
68
69     return total;
70
71 } // end function totalAnnualSales

```

Hàm tiện ích **private** phục vụ hàm **printAnnualSales**; đóng gói thao tác trên mảng **sales**.



**fig06\_11.cpp**  
(1 of 1)

Chuỗi gọi hàm đơn giản;  
logic chương trình được đóng gói trong các  
hàm thành viên.

```
1 // Fig. 6.11: fig06_11.cpp
2 // Demonstrating a utility function.
3 // Compile this program with salesp.cpp
4
5 // include SalesPerson class definition from salesp.h
6 #include "salesp.h"
7
8 int main()
9 {
10     SalesPerson s;           // create SalesPerson object s
11
12     s.getSalesFromUser();    // note simple sequential code; no
13     s.printAnnualSales();    // control structures in main
14
15     return 0;
16
17 } // end main
```

**fig06\_11.cpp**  
**output (1 of 1)**

```
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
```

## 6.10 Khởi tạo các đối tượng: Constructor

- Constructors
  - khởi tạo các thành viên dữ liệu
    - hoặc có thể gán trị cho các thành viên dữ liệu sau
  - trùng tên với tên lớp
  - không có kiểu trả về

- Các giá trị khởi tạo – Initializers

- được truyền dưới dạng đối số cho constructor
- khi khai báo biến: đặt trong cặp ngoặc đơn trước dấu chấm phẩy

*Class-type ObjectName( value1, value2, ... );*

```
class Time {  
public:  
    Time( int, int, int );  
...  
}; // end class Time  
...  
int main()  
{  
    Time t( 27, 74, 99 );  
...  
}
```

## 6.11 Sử dụng các đối số mặc định với constructor

- có thể chỉ định các đối số mặc định
  - tương tự đối số mặc định của hàm thông thường
- constructor mặc định:
  - có thể gọi không cần tham số
    - `Time t;`
  - Tất cả các đối số là mặc định HOẶC thực sự không nhận tham số
    - `Time (int = 0, int = 0, int = 0);`
    - hoặc
    - `Time ();`
  - mỗi lớp chỉ được có tối đa một constructor mặc định

```
1 // Fig. 6.12: time2.h
2 // Declaration of class Time.
3 // Member functions defined in time2.cpp.
4
5 // prevent multiple inclusions of header file
6 #ifndef TIME2_H
7 #define TIME2_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(); // print universal-time format
16     void printStandard(); // print standard-time format
17
18 private:
19     int hour; // 0 - 23 (24-hour clock format)
20     int minute; // 0 - 59
21     int second; // 0 - 59
22
23 }; // end class Time
24
25 #endif
```

Default constructor chỉ định giá trị mặc định cho mọi đối số.

## time2.cpp (1 of 2)

```
1 // Fig. 6.13: time2.cpp
2 // Member-function definitions for class Time.
3 #include <iostream>
4
5 using std::cout;
6
7 #include <iomanip>
8
9 using std::setfill;
10 using std::setw;
11
12 // include definition of class Time from time2.h
13 #include "time2.h"
14
15 // Time constructor initializes each data member to zero;
16 // ensures all Time objects start in a consistent state
17 Time::Time( int hr, int min, int sec )
18 {
19     setTime( hr, min, sec ); // validate and set time
20
21 } // end Time constructor
22
```

Constructor gọi **setTime** để kiểm tra các giá trị được truyền vào (hoặc mặc định).

## time2.cpp (2 of 2)

```
23 // set new Time value using universal time, perform validity
24 // checks on the data values and set invalid values to zero
25 void Time::setTime( int h, int m, int s )
26 {
27     hour = ( h >= 0 && h < 24 ) ? h : 0;
28     minute = ( m >= 0 && m < 60 ) ? m : 0;
29     second = ( s >= 0 && s < 60 ) ? s : 0;
30
31 } // end function setTime
32
33 // print Time in universal format
34 void Time::printUniversal()
35 {
36     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
37         << setw( 2 ) << minute << ":"
38         << setw( 2 ) << second;
39
40 } // end function printUniversal
41
42 // print Time in standard format
43 void Time::printStandard()
44 {
45     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
46         << ":" << setfill( '0' ) << setw( 2 ) << minute
47         << ":" << setw( 2 ) << second
48         << ( hour < 12 ? " AM" : " PM" );
49
50 } // end function printStandard
```

fig06\_14.cpp  
(1 of 2)

```
1 // Fig. 6.14: fig06_14.cpp
2 // Demonstrating a default constructor for class Time.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // include definition of class Time from time2.h
9 #include "time2.h"
10
11 int main()
12 {
13     Time t1;           // all arguments defaulted
14     Time t2( 2 );      // minute and second defaulted
15     Time t3( 21, 34 ); // second defaulted
16     Time t4( 12, 25, 42 ); // all values specified
17     Time t5( 27, 74, 99 ); // all bad values specified
18
19     cout << "Constructed with:\n\n"
20          << "all default arguments:\n ";
21     t1.printUniversal(); // 00:00:00
22     cout << "\n ";
23     t1.printStandard(); // 12:00:00 AM
24 }
```

Khởi tạo các đối tượng **Time** sử dụng các tham số mặc định.

Khởi tạo đối tượng **Time** với các giá trị không hợp lệ; khâu kiểm tra tính hợp lệ sẽ gán các giá trị về 0.



fig06\_14.cpp  
(2 of 2)

```
25 cout << "\n\nhour specified; default minute and second:\n ";
26 t2.printUniversal(); // 02:00:00
27 cout << "\n ";
28 t2.printStandard(); // 2:00:00 AM
29
30 cout << "\n\nhour and minute specified; default second:\n ";
31 t3.printUniversal(); // 21:34:00
32 cout << "\n ";
33 t3.printStandard(); // 9:34:00 PM
34
35 cout << "\n\nhour, minute, and second specified:\n ";
36 t4.printUniversal(); // 12:25:42
37 cout << "\n ";
38 t4.printStandard(); // 12:25:42 PM
39
40 cout << "\n\nall invalid values specified:\n ";
41 t5.printUniversal(); // 00:00:00
42 cout << "\n ";
43 t5.printStandard(); // 12:00:00 AM
44 cout << endl;
45
46 return 0;
47
48 } // end main
```

t5 được xây dựng bằng các  
đối số không hợp lệ, các giá  
trị được gán về 0.

## 6.12 Destructor – hàm hủy

- Destructor – hàm thành viên tự hủy của đối tượng
  - hàm thành viên đặc biệt
  - trùng tên với tên lớp
    - bắt đầu bằng dấu ngã (~)
  - không nhận đối số
  - không có giá trị trả về
  - không thể bị gọi chồng
  - thực hiện việc dọn dẹp
    - trước khi hệ thống lấy lại phần bộ nhớ của đối tượng
      - tái sử dụng cho đối tượng mới
  - nếu không có destructor được định nghĩa tường minh
    - trình biên dịch tự tạo destructor "rỗng" – không làm gì hết

## 6.13 Khi nào Constructor và Destructor được gọi

- các constructor và destructor
  - được gọi ngầm bởi trình biên dịch
- thứ tự gọi hàm
  - phụ thuộc vào thứ tự thực thi chương trình
    - khi chương trình vào và ra khỏi phạm vi của các đối tượng
  - các đối tượng cũng là các biến thông thường,
    - biến được khởi tạo – constructor được gọi – tại thời điểm bắt đầu tồn tại / phạm vi
    - biến bị hủy – destructor được gọi – khi kết thúc sự tồn tại / ra khỏi phạm vi
  - thông thường, các lời gọi destructor theo thứ tự ngược lại với thứ tự gọi các constructor

## 6.13 Khi nào Constructor và Destructor được gọi

- Thứ tự các lời gọi constructor, destructor
  - đối với các đối tượng/biến phạm vi toàn cục (global scope objects)
    - Constructor
      - được gọi trước mọi hàm khác (kể cả **main**)
    - Destructor
      - được gọi khi **main** kết thúc (hoặc khi hàm **exit** được gọi)
      - không được gọi nếu chương trình kết thúc bằng hàm **abort**

## 6.13 Khi nào Constructor và Destructor được gọi

- Thứ tự các lời gọi constructor, destructor
  - đối với các đối tượng/biến địa phương (automatic local objects)
    - Constructor
      - được gọi khi đối tượng được định nghĩa
        - mỗi khi chương trình vào phạm vi của đối tượng
    - Destructor
      - được gọi khi đối tượng ra khỏi phạm vi
        - chương trình ra khỏi khối nơi đối tượng được định nghĩa
      - không được gọi nếu chương trình kết thúc bằng **exit** hay **abort**

## 6.13 Khi nào Constructor và Destructor được gọi

- Thứ tự các lời gọi constructor, destructor
  - các đối tượng tĩnh địa phương (**static** local objects)
    - Constructor
      - đúng một lần
      - khi chương trình chạy đến chỗ đối tượng được định nghĩa
    - Destructor
      - khi hàm **main** kết thúc hoặc khi hàm **exit** được gọi
      - không được gọi nếu chương trình kết thúc bằng hàm **abort**

```
1 // Fig. 6.15: create.h
2 // Definition of class CreateAndDestroy.
3 // Member functions defined in create.cpp.
4 #ifndef CREATE_H
5 #define CREATE_H
6
7 class CreateAndDestroy {
8
9 public:
10     CreateAndDestroy( int, char * ); // constructor
11     ~CreateAndDestroy();             // destructor
12
13 private:
14     int objectID;
15     char *message;
16
17 }; // end class CreateAndDestroy
18
19 #endif
```

Các hàm thành viên  
constructor và destructor

Các thành viên **private**  
để minh họa thứ tự các lời gọi  
constructor và destructor

## create.cpp (1 of 2)

```
1 // Fig. 6.16: create.cpp
2 // Member-function definitions for class CreateAndDestroy
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // include CreateAndDestroy class definition from create.h
9 #include "create.h"
10
11 // constructor
12 CreateAndDestroy::CreateAndDestroy(
13     int objectNumber, char *messagePtr )
14 {
15     objectID = objectNumber;
16     message = messagePtr;
17
18     cout << "Object " << objectID << " constructor runs "
19         << message << endl;
20
21 } // end CreateAndDestroy constructor
22
```

Output message để thể hiện thời gian của các lời gọi hàm constructor.



Output message để thể hiện thời gian  
của các lời gọi hàm destructor

create.cpp (2 of 2)

```
23 // destructor
24 CreateAndDestroy::~~CreateAndDestroy()
25 {
26     // the following line is for pedagogic purposes only
27     cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
28
29     cout << "Object " << objectID << "    destructor runs    "
30           << message << endl;
31
32 } // end ~CreateAndDestroy destructor
```

fig06\_17.cpp  
(1 of 2)

```
1 // Fig. 6.17: fig06_17.cpp
2 // Demonstrating the order in which constructors and
3 // destructors are called.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 // include CreateAndDestroy class definition from create.h
10 #include "create.h"
11
12 void create( void ); // prototype
13
14 // global object
15 CreateAndDestroy first( 1, "(global before main)" );
16
17 int main()
18 {
19     cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
20
21     CreateAndDestroy second( 2, "(local automatic in main)" );
22
23     static CreateAndDestroy third( 3, "(local static in main)" );
24
25     create(); // call function to create objects
26
```

tạo đối tượng có phạm vi toàn cục

Tạo đối tượng tự động địa phương.

Tạo đối tượng địa phương static.

Tạo các đối tượng tự động địa phương.

```
cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;

CreateAndDestroy fourth( 4, "(local automatic in main)" );

cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;

return 0;

} // end main

// function to create objects
void create( void )
{
    cout << "\nCREATE FUNCTION: EXECUTION BEGINS" << endl;

    CreateAndDestroy fifth( 5, "(local automatic in create)" );

    static CreateAndDestroy sixth(
        6, "(local static in create)" );

    CreateAndDestroy seventh(
        7, "(local automatic in create)" );

    cout << "\nCREATE FUNCTION: EXECUTION ENDS\" << endl;

} // end function create
```

Tạo đối tượng tự động địa phương.

Tạo đối tượng tự động địa phương bên trong hàm.

Tạo đối tượng địa phương **static** bên trong hàm.

Tạo đối tượng tự động địa phương bên trong hàm.

fig06\_17.cpp  
output (1 of 1)

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)

đối tượng **static** địa  
đối tượng toàn cục được tạo  
đối tượng **static** địa  
phương được tạo tại lời gọi  
hàm đầu tiên và hủy sau khi  
hàm **main** kết thúc.

## 6.14 Sử dụng các hàm truy nhập

- Set functions – các hàm ghi
  - kiểm tra tính hợp lệ trước khi sửa đổi dữ liệu **private**
  - thông báo nếu các giá trị là không hợp lệ
  - thông báo qua các giá trị trả về
- Get functions – các hàm đọc
  - các hàm truy vấn – “Query” functions
  - quản lý định dạng của dữ liệu trả về

```
1 // Fig. 6.18: time3.h
2 // Declaration of class Time.
3 // Member functions defined in time3.cpp
4
5 // prevent multiple inclusions of header file
6 #ifndef TIME3_H
7 #define TIME3_H
8
9 class Time {
10
11 public:
12     Time( int = 0, int = 0, int = 0 ); // default constructor
13
14     // set functions
15     void setTime( int, int, int ); // set hour, minute, second
16     void setHour( int ); // set hour
17     void setMinute( int ); // set minute
18     void setSecond( int ); // set second
19
20     // get functions
21     int getHour(); // return hour
22     int getMinute(); // return minute
23     int getSecond(); // return second
24
```

Các hàm ghi

các hàm đọc

```
25     void printUniversal(); // output universal-time format
26     void printStandard(); // output standard-time format
27
28 private:
29     int hour;                // 0 - 23 (24-hour clock format)
30     int minute;             // 0 - 59
31     int second;             // 0 - 59
32
33 }; // end clas Time
34
35 #endif
```

**time3.h (2 of 2)**

## time3.cpp (1 of 4)

```
1  // Fig. 6.19: time3.cpp
2  // Member-function definitions for Time class.
3  #include <iostream>
4
5  using std::cout;
6
7  #include <iomanip>
8
9  using std::setfill;
10 using std::setw;
11
12 // include definition of class Time from time3.h
13 #include "time3.h"
14
15 // constructor function to initialize private data;
16 // calls member function setTime to set variables;
17 // default values are 0 (see class definition)
18 Time::Time( int hr, int min, int sec )
19 {
20     setTime( hr, min, sec );
21
22 } // end Time constructor
23
```



```
24 // set hour, minute and second values
25 void Time::setTime( int h, int m, int s )
26 {
27     setHour( h );
28     setMinute( m );
29     setSecond( s );
30
31 } // end function setTime
32
33 // set hour value
34 void Time::setHour( int h )
35 {
36     hour = ( h >= 0 && h < 24 ) ? h : 0;
37
38 } // end function setHour
39
40 // set minute value
41 void Time::setMinute( int m )
42 {
43     minute = ( m >= 0 && m < 60 ) ? m : 0;
44
45 } // end function setMinute
46
```

Gọi các hàm set để kiểm tra tính hợp lệ.

Các hàm set kiểm tra tính hợp lệ trước khi sửa đổi dữ liệu.

Các hàm set kiểm tra tính hợp lệ trước khi sửa đổi dữ liệu.

time3.cpp (3 of 4)

```
47 // set second value
48 void Time::setSecond( int s )
49 {
50     second = ( s >= 0 && s < 60 ) ? s : 0;
51
52 } // end function setSecond
53
54 // return hour value
55 int Time::getHour()
56 {
57     return hour;
58
59 } // end function getHour
60
61 // return minute value
62 int Time::getMinute()
63 {
64     return minute;
65
66 } // end function getMinute
67
```

Các hàm get cho client đọc dữ liệu



Hàm get cho client đọc dữ liệu

```
68 // return second value
69 int Time::getSecond()
70 {
71     return second;
72 }
73 // end function getSecond
74
75 // print Time in universal format
76 void Time::printUniversal()
77 {
78     cout << setfill( '0' ) << setw( 2 ) << hour << ":"
79         << setw( 2 ) << minute << ":"
80         << setw( 2 ) << second;
81 }
82 // end function printUniversal
83
84 // print Time in standard format
85 void Time::printStandard()
86 {
87     cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
88         << ":" << setfill( '0' ) << setw( 2 ) << minute
89         << ":" << setw( 2 ) << second
90         << ( hour < 12 ? " AM" : " PM" );
91 }
92 // end function printStandard
```

fig06\_20.cpp  
(1 of 3)

```
1 // Fig. 6.20: fig06_20.cpp
2 // Demonstrating the Time class set and get functions
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 // include definition of class Time from time3.h
9 #include "time3.h"
10
11 void incrementMinutes( Time &, const int ); // prototype
12
13 int main()
14 {
15     Time t; // create Time object
16
17     // set time using individual set functions
18     t.setHour( 17 ); // set hour to valid value
19     t.setMinute( 34 ); // set minute to valid value
20     t.setSecond( 25 ); // set second to valid value
21 }
```

Gọi các hàm set để gán các giá trị hợp lệ.

```
22 // use get functions to obtain hour, minute and second
```

```
23 cout << "Result of setting all valid values:\n"
```

```
24     << "   Hour: " << t.getHour()
```

```
25     << "   Minute: " << t.getMinute()
```

```
26     << "   Second: " << t.getSecond();
```

Cố dùng các hàm set để gán các giá trị không hợp lệ.

```
27
```

```
28 // set time using individual set functions
```

```
29 t.setHour( 234 );    // invalid hour set to 0
```

```
30 t.setMinute( 43 );  // set minute to valid value
```

```
31 t.setSecond( 6373 ); // invalid second set to 0
```

các giá trị không hợp lệ làm các data member bị gán về 0.

```
32
```

```
33 // display hour, minute and second after setting
```

```
34 // invalid hour and second values
```

```
35 cout << "\n\nResult of attempting to set invalid hour and"
```

```
36     << "   second:\n   Hour: " << t.getHour()
```

```
37     << "   Minute: " << t.getMinute()
```

```
38     << "   Second: " << t.getSecond() << "\n\n";
```

Sửa đổi data member bằng hàm setTime.

```
39
```

```
40 t.setTime( 11, 58, 0 );    // set time
```

```
41 incrementMinutes( t, 3 );  // increment t's minute by 3
```

```
42
```

```
43 return 0;
```

```
44
```

```
45 } // end main
```

```
46
```

fig06\_20.cpp  
(3 of 3)

```
47 // add specified number of minutes to a Time object
48 void incrementMinutes( Time &tt, const int count )
49 {
50     cout << "Incrementing minute " << count
51         << " times:\nStart time: ";
52     tt.printStandard();
53
54     for ( int i = 0; i < count; i++ ) {
55         tt.setMinute( ( tt.getMinute() + 1 ) % 60 );
56
57         if ( tt.getMinute() == 0 )
58             tt.setHour( ( tt.getHour() + 1 ) % 24 );
59
60         cout << "\nminute + 1: ";
61         tt.printStandard();
62
63     } // end for
64
65     cout << endl;
66
67 } // end function increment
```

Dùng các hàm get để đọc và các hàm set để sửa dữ liệu.

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

Cố gắng gán các giá trị không hợp lệ cho các thành viên dữ liệu, kết quả là thông báo lỗi và các thành viên bị gán về 0.

## 6.15 Phép gán mặc định

- Gán đối tượng cho đối tượng
  - Phép gán (=)
    - có thể gán một đối tượng cho một đối tượng khác thuộc cùng kiểu
    - Mặc định: gán theo từng thành viên (memberwise assignment)
      - Mỗi thành viên của đối tượng vế phải được gán cho thành viên tương ứng tại vế trái
- được ngầm thực hiện khi
  - truyền tham số là đối tượng
  - trả về đối tượng
- Đối tượng có thể được truyền làm tham số cho hàm
  - Đối tượng có thể được hàm trả về
  - Mặc định: pass-by-value
    - Bản sao của đối tượng được truyền, trả về
      - sử dụng 'copy constructor'
        - sao chép các giá trị gốc vào đối tượng mới

**fig06\_24.cpp**  
(1 of 3)

```
1 // Fig. 6.24: fig06_24.cpp
2 // Demonstrating that class objects can be assigned
3 // to each other using default memberwise assignment.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8
9 // class Date definition
10 class Date {
11
12 public:
13     Date( int = 1, int = 1, int = 1990 ); // default constructor
14     void print();
15
16 private:
17     int month;
18     int day;
19     int year;
20
21 }; // end class Date
22
```



**fig06\_24.cpp**  
(2 of 3)

```
23 // Date constructor with no range checking
24 Date::Date( int m, int d, int y )
25 {
26     month = m;
27     day = d;
28     year = y;
29
30 } // end Date constructor
31
32 // print Date in the format mm-dd-yyyy
33 void Date::print()
34 {
35     cout << month << '-' << day << '-' << year;
36
37 } // end function print
38
39 int main()
40 {
41     Date date1( 7, 4, 2002 );
42     Date date2; // date2 defaults to 1/1/1990
43
```

```
44  cout << "date1 = ";
45  date1.print();
46  cout << "\ndate2 = ";
47  date2.print();
48
49  date2 = date1;    // default memberwise assignment
50
51  cout << "\n\nAfter default memberwise assignment, date2 = ";
52  date2.print();
53  cout << endl;
54
55  return 0;
56
57 } // end main
```

phép gán mặc định gán từng thành viên của **date1** cho thành viên tương ứng của **date2**.

fig06\_24.cpp  
(3 of 3)

fig06\_24.cpp  
output (1 of 1)

```
date1 = 7-4-2002
date2 = 1-1-1990
```

```
After default memberwise assignment, date2 = 7-4-2002
```