



# Introduction to C++ Programming

---

## Lớp (Class)



# **Cấu trúc - Structures**

# Định nghĩa

- Kiểu dữ liệu cấu trúc - Structures
  - để lưu các dữ liệu là tập các kiểu dữ liệu khác nhau

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

Structure tag

Structure members

- Lưu ý
  - trong cùng **struct**: phải có tên khác nhau
  - với **structs** khác nhau: có thể dùng chung tên
  - **Định nghĩa struct** kết thúc bằng dấu chấm phẩy

# Định nghĩa

- Khai báo biến kiểu **struct**
  - khai báo giống như các kiểu biến khác
  - Ví dụ:
    - `Time timeObject; //biến bình thường`
    - `Time timeArray[ 10 ]; //mảng`
    - `Time *timePtr; //con trỏ`

# Truy cập các thành phần kiểu cấu trúc

- Toán tử truy cập

- Toán tử (.) cho các thành phần cấu trúc với kiểu biến thường
- Toán tử (->) cho các thành phần cấu trúc với kiểu POINTER
- Hiển thị thành phần **hour** của **timeObject**:

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

hoặc

```
timePtr = &timeObject;
```

```
cout << timePtr->hour;
```

- **timePtr->hour** giống như ( **\*timePtr** ).hour

- Cần có dấu ngoặc đơn () vì
  - \* có mức ưu tiên thấp hơn .

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
19 }; // end struct Time
20
21 void printUniversal( const Time & ); // prototype
22 void printStandard( const Time & ); // prototype
23
```

Define structure type **Time**  
with three integer members.

Pass references to constant  
**Time** objects to eliminate  
copying overhead.



fig06\_01.cpp  
(2 of 3)

Use dot operator to initialize  
structure members.

Direct access to data allows  
assignment of bad values.

```
24 int main()
25 {
26     Time dinnerTime;           //
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)

```

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

```

Use parameterized stream manipulator **setfill**.

Use dot operator to access data members.

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



## Cấu trúc tự trỏ

- có ít nhất 1 thành phần là con trỏ với kiểu là struct đang định nghĩa

```
struct listNum{  
    int num;  
    struct listNum *next;    // Trỏ đến node kế tiếp  
};
```

- Ứng dụng:
  - danh sách
  - ngăn xếp
  - hàng đợi

# Ví dụ

```
typedef struct{  
    char name[25];           // Tên nhân viên  
    int age;                  // Tuổi nhân viên  
    float salary;             // Lương nhân viên  
} Employee;  
  
struct simple{  
    Employee employee;        // Dữ liệu của node  
    struct simple *next;      // Trỏ đến node kế tiếp  
};  
typedef struct simple SimpleNode;
```

# Nhận xét về `struct`

- C-style structures
  - Không có giao diện (“interface”)
    - nếu có thay đổi, toàn bộ chương trình sử dụng **struct** tương ứng phải thay đổi theo
  - Không in được giống như biến
    - Phải in từng thành phần một
  - Không thể so sánh như biến
    - Phải so sánh từng thành phần một

## Lớp - class

Để định nghĩa một lớp trong C++, ta dùng từ khóa **class** với cú pháp:

```
class <Tên lớp>{  
private:  
<Khai báo các thành phần riêng>  
protected:  
<Khai báo các thành phần được bảo vệ>  
public:  
<Khai báo các thành phần công cộng>  
};
```

## Lớp - class

### Thành phần

- thuộc tính (data members)
- phương thức (member functions)
- Định nghĩa dùng từ khoá class

### Phạm vi truy cập

- public: truy cập ở tất cả mọi nơi
- Private: Chỉ phương thức của lớp mới truy cập được
- Protected: tương tự private

## Hàm tạo

- Hàm tạo

- Hàm thành phần đặc biệt
  - Khởi tạo cho các thành phần dữ liệu
  - Cùng tên với lớp
- Được gọi TỰ ĐỘNG khi tạo đối tượng
- Có thể có nhiều hàm tạo
  - Chồng hàm
- Không có kiểu trả về

```

1  class Time {
2
3  public:
4      Time();
5      void setTime(int hour, int minute, int second);
6      void printUniversal();
7      void printStandard();
8
9  private:
10     int hour;           // 0 - 23 (định dạng 24-giờ)
11     int minute;         // 0 - 59
12     int second;         // 0 - 59
13
14 }; // end class Time

```

Định nghĩa hàm thành phần **public**.

Thân class bắt đầu bằng dấu {.

Hàm tạo có cùng tên với lớp, **Time**, không có giá trị trả về.

Kết thúc định nghĩa bằng dấu ;.

Thân hàm kết thúc bằng dấu }

Định nghĩa lớp Time

# Đối tượng của lớp

- Đối tượng
  - Sau khi định nghĩa lớp
    - Kiểu lớp mới được tạo
    - Có thể khai báo Đối tượng, mảng, con trỏ và tham chiếu
  - Ví dụ:

Tên Class thành một kiểu mới.

```
Time sunset; // đối tượng của kiểu Time
Time arrayOfTimes[ 5 ]; // Mảng các đối tượng kiểu Time
Time *pointerToTime; // Con trỏ trỏ tới đối tượng kiểu Time
Time &dinnerTime = sunset; // Tham chiếu tới đối tượng kiểu Time
```



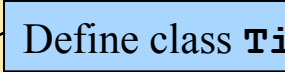
# Hàm thành phần

- Định nghĩa các hàm thành phần bên ngoài phạm vi lớp
  - Toán tử phạm vi (`::`)
    - “Buộc” hàm thành phần vào tên lớp
    - Phân biệt các hàm ở các lớp khác nhau
    - Lớp khác nhau có thể có hàm thành phần cùng tên
  - Cú pháp định nghĩa hàm thành viên

```
ReturnType ClassName::MemberFunctionName ( ) {  
    ...  
}
```
  - Không thay đổi với hàm **public** hay **private**
- Hàm thành phần được định nghĩa trong lớp
  - Không cần toán tử phạm vi, tên lớp
  - Trình biên dịch nhận định là hàm **inline**
    - Ngoài lớp, inline được xác định thông qua từ khoá **inline**

fig06\_03.cpp  
(1 of 5)

```
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
```



# Hàm huỷ

- Hàm huỷ
  - Cùng tên với lớp
    - Bắt đầu bằng ký hiệu ~
  - Không có đối số
  - Không chồng hàm được
  - Thực hiện “dọn dẹp” ( “termination housekeeping”)
    - ví dụ: đóng file đã mở, thu hồi bộ nhớ đã cấp phát...

## Gọi hàm huỷ và hàm tạo

- Constructors và destructors
  - Được gọi (ngầm) bởi trình biên dịch
- Thứ tự gọi hàm
  - Tùy vào thứ tự thực hiện
    - Khi bắt đầu và kết thúc phạm vi của các đối tượng
  - Thông thường, thứ tự gọi hàm huỷ ngược với hàm tạo

# Gọi hàm huỷ và hàm tạo

- Thứ tự gọi hàm tạo và hàm huỷ
  - Đối tượng có phạm vi toàn cục
    - Hàm tạo
      - được gọi trước tất cả các hàm (bao gồm cả **main**)
    - Hàm huỷ
      - Khi kết thúc **main** (hoặc khi gọi tới hàm **exit**)
      - không được gọi nếu chương trình kết thúc bằng **abort**
  - Đối tượng có phạm vi địa phương
    - Hàm tạo
      - Khi đối tượng được tạo
        - Trong block và đã định nghĩa đối tượng
    - Hàm huỷ
      - Khi ra khỏi phạm vi của đối tượng
        - Thoát khỏi block mà đã định nghĩa đối tượng
      - không được gọi nếu chương trình kết thúc do **exit** hoặc **abort**

# Gọi hàm huỷ và hàm tạo

## Thứ tự gọi hàm tạo và hàm huỷ

- **đối tượng static** địa phương
  - Hàm tạo
    - Gọi duy nhất 1 lần
    - Khi gặp câu lệnh khai báo đối tượng
  - Hàm huỷ
    - Khi kết thúc hàm **main** hoặc gọi tới hàm **exit**
    - Không được gọi nếu chương trình kết thúc bởi **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();
12
13 private:
14     int objectID;
15     char *message;
16
17 }; // end class CreateAndDestroy
18
19 #endif
```

Constructor and destructor  
member functions.

**private** members to show  
order of constructor,  
destructor function calls.

create.h (1 of 1)

```
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
```

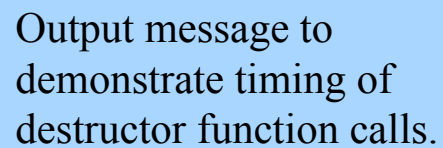
Output message to demonstrate timing of constructor function calls.

create.cpp (1 of 2)



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

Output message to demonstrate timing of destructor function calls.



```
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\n";
20
21     CreateAndDestroy second( 2, "(local automatic in main)" );
22
23     static CreateAndDestroy third(
24         3, "(local static in main)" );
25 }
```

Create variable with global scope.

Create local automatic object.

Create **static** local object.

fig06\_17.cpp  
(1 of 3)

```

26  create(); // call function to create objects
27
28  cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
29
30  CreateAndDestroy fourth( " );
31
32  cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
33
34  return 0;
35
36 } // end main
37
38 // function to create objects
39 void create( void )
40 {
41     cout << "\nCREATE FUNCTION" << endl;
42
43     CreateAndDestroy fifth( 5, " " );
44
45     static CreateAndDestroy sixth( 6, "(local static in create)" );
46
47     CreateAndDestroy seventh( 7, "(local automatic in create)" );
48
49
50

```

Annotations for the code above:

- Annotation 1 (points to line 26): Create local automatic objects.
- Annotation 2 (points to line 32): Create local automatic object.
- Annotation 3 (points to line 41): Create local automatic object in function.
- Annotation 4 (points to line 43): Create **static** local object in function.
- Annotation 5 (points to line 45): Create local automatic object in function.

fig06\_17.cpp  
(2 of 3)

```
51     cout << "\nCREATE FUNCTION: EXECUTION ENDS\n" << endl;  
52  
53 } // end function create
```

2

8

fig06\_17.cpp  
(3 of 3)

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)

Local static object exists  
Global object constructed  
Local static object  
Local **static** object  
constructed on first function  
call and destroyed after **main**  
execution ends.

## Ưu điểm của dùng lớp

- Ưu điểm của dùng class
  - Chương trình đơn giản
  - Giao diện
    - Ăn cài đặt chi tiết
  - Tái sử dụng phần mềm
    - Thành phần (aggregation)
      - Đối tượng được khai báo như một thành phần của lớp khác
    - Kế thừa
      - Lớp mới được tạo từ lớp đã có

# Phạm vi của lớp & truy cập tới các thành phần của lớp

- Phạm vi của lớp
  - Thành phần dữ liệu, thành phần hàm
  - Trong phạm vi lớp
    - Thành phần của lớp
      - có thể truy cập trực tiếp bởi các hàm thành phần thông qua tên
  - Ngoài phạm vi lớp
    - Truy cập thông qua handles
      - Tên đối tượng, tham chiếu tới đối tượng, con trỏ đối tượng
- Phạm vi file
  - Hàm không phải thành phần của lớp

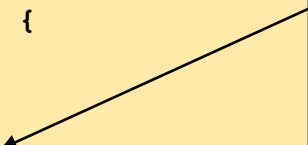
## Phạm vi của lớp & truy cập tới các thành phần của lớp

- Phép toán truy cập các thành phần của lớp
  - tương tự như với **structs**
  - Dùng toán tử (.) cho
    - đối tượng
    - Tham chiếu tới đối tượng
  - Dùng toán tử (->) cho
    - con trỏ



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
```



Data member **x** **public** to  
illustrate class member access  
operators; typically data  
members **private**.

fig06\_04.cpp  
(2 of 2)

fig06\_04.cpp  
output (1 of 1)

```
23 int main()
24 {
25     Count counter;                // create counter object
26     Count *counterPtr = &counter; // create pointer to counter
27     Count &counterRef = counter;  // Use dot member selection
28                                   // operator for counter
29     cout << "Assign 1 to x and print\n";
30     counter.x = 1;                // assign 1 to
31     counter.print();              // call member
32                                   // Use dot member selection
33                                   // operator for counterRef
34     cout << "Assign 2 to x and print using reference to object.\n";
35     counterRef.x = 2;             // assign 2 to
36     counterRef.print();           // call member
37                                   // Use arrow member selection
38     cout << "Assign 3 to x and print using pointer to object.\n";
39     counterPtr->x = 3;             // assign 3 to data member x
40     counterPtr->print();           // call member function print
41
42     - counter.print();
43
44     return 0;
45 }
```

Use dot member selection operator for **counter** object.

Use dot member selection operator for **counterRef** reference to object.

Use arrow member selection operator for **counterPtr** pointer to object.

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

## Tách biệt giao diện và cài đặt

- Tách biệt giao diện và cài đặt
  - Ưu
    - dễ chỉnh sửa chương trình
  - Nhược
    - Phải dùng header files

# Tách biệt giao diện và cài đặt

- Header files
  - Định nghĩa lớp và khuôn mẫu hàm
  - Được include ở các file dùng lớp
    - **#include**
  - Phần mở rộng là **.h**
- Source-code files
  - Định nghĩa các hàm thành phần



time1.h (1 of 1)

```

1  // Fig. 6.5: time1.h
2  // Declaration of class Time.
3  // Member functions are defined in t1.cpp
4
5  // prevent multiple inclusions of header file
6  #ifndef TIME1_H //if not define - nếu chưa định nghĩa TIME1_H
7  #define TIME1_H
8
9  // Time abstract class
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

```

Preprocessor code to prevent multiple inclusions.

Code between these

"If not defined" defines

Naming convention: header file name with underscore replacing period.

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' has no member named 'hour'
16
17     // error: 'Time::minute' is not accessible
18     cout << "minute = " << t.minute;
19
20
21     return 0;
22
23 } // end main
```

Recall data member **hour** is **private**; attempts to access **private** members results in error.

Data member **minute** also **private**; attempts to access **private** members produces error.

## Điều khiển truy cập tới các thành phần

- Truy cập tới các thành phần của lớp
  - Mặc định là **private**
  - Trừ khi thiết lập **private**, **public**, **protected**
- Truy cập tới các thành phần của **struct**
  - Mặc định là **public**
  - Trừ khi thiết lập là **private**, **public**, **protected**
- Truy cập tới các dữ liệu **private** của lớp
  - Điều khiển với các hàm truy cập (accessor methods)
    - Hàm get - Get function
      - Đọc dữ liệu **private**
    - Hàm set - Set function
      - Chỉnh sửa(gán) dữ liệu **private**

```
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
13     void printAnnualSales(); // summarize
14
15 private:
16     double totalAnnualSales(); // utility function
17     double sales[ 12 ];        // 12 monthly sales figures
18
19 }; // end class SalesPerson
20
21 #endif
```

Set access function performs validity checks.

**private** utility function.





```
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
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;
```

Set access function performs  
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
```

**private** utility function to help function **printAnnualSales**; encapsulates logic of manipulating **sales** array.

fig06\_11.cpp  
(1 of 1)

```
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
11
12     s.getSalesFromUser();    // note simple sequential code, no
13     s.printAnnualSales();    // control structures in main
14
15     return 0;
16
17 }
```

Simple sequence of member function calls; logic encapsulated in member functions.

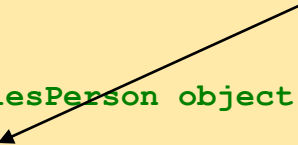




fig06\_11.cpp  
output (1 of 1)

```
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
```

## Dùng hàm *Set* & *Get*

- **Set** functions

- Thực hiện kiểm tra tính hợp lệ trước khi thay đổi dữ liệu **private**
- Thông báo nếu giá trị không hợp lệ (thông qua giá trị trả về)

- **Get** functions

- Hàm truy vấn “Query”
- Xác định khuôn mẫu 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
```

 Set functions. Get functions.



```
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
```





```
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
```

Call set functions to perform validity checking.

Set functions perform validity checks before modifying data.

```
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
```

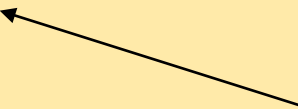
Set function performs validity checks before modifying data.

Get functions allow client to read data.



time3.cpp (4 of 4)

```
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
```

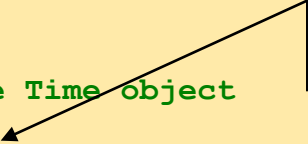


Get function allows client to read data.

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 }
```

Invoke set functions to set  
valid values.





cpp

(2 01 3)

Attempt to set invalid values using set functions.

Invalid values result in setting data members to 0.

Modify data members using function **setTime**.

```
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();
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
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";
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

Using get functions to read data and set functions to modify data.

```
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 incrementMinutes
```



fig06\_20.cpp  
output (1 of 1)

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

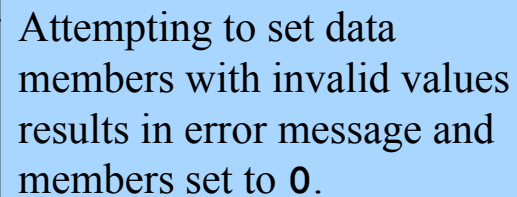
Incrementing minute 3 times:

Start time: 11:58:00 AM

minute + 1: 11:59:00 AM

minute + 1: 12:00:00 PM

minute + 1: 12:01:00 PM



Attempting to set data  
members with invalid values  
results in error message and  
members set to 0.



Viết chương trình sử dụng class để

1. **Ghi** n nhân viên (name, age, salary, ID) vào file (tên file nhập từ bàn phím)
2. **đọc** danh sách nhân viên từ file đã nhập và ~~in~~ ra màn hình
3. **tìm** nhân viên với ID = 102 trong file và **đổi** salary thành 4.000.000
4. **Xoá** các nhân viên có age < 18 khỏi file
5. **Chèn** thêm 1 nhân viên vào file với thông tin nhân viên được ~~nhập~~ từ bàn phím.