Ngôn ngữ lập trình

Bài 4:
Cấu trúc và lớp

Giảng viên: Lê Nguyễn Tuấn Thành
Email: thanhlnt@tlu.edu.vn

Bộ Môn Công Nghệ Phần Mềm – Khoa CNTT
Trường Đại Học Thủy Lợi

Nội dung

1. Kiểu cấu trúc
2. Kiểu lớp
3. Hàm tạo & Hàm hủy

Bài giảng có sử dụng hình vẽ trong cuốn sách "*Absolute C++*. W. Savitch, Addison Wesley, 2002"

1. KIỂU CẤU TRÚC

(Struct)

Mục tiêu bài học

- Các kiểu cấu trúc (structure)
- Sử dụng cấu trúc như đối số của hàm
- Khởi tạo cấu trúc

Cấu trúc

- Kiểu dữ liệu tổng hợp thứ hai (sau mảng): *struct*
- Nhớ lại: kiểu dữ liệu tổng hợp nghĩa là "*nhóm dữ liệu lại với nhau*" (grouping)
  - Mảng (array): tập hợp các giá trị CÙNG KiỂU
  - Cấu trúc (structure): tập hợp các giá trị KHÁC KiỂU
- Được coi như một đối tượng đơn, giống như mảng
- Điểm khác nhau chính: phải ĐỊNH NGHĨA cấu trúc TRƯỚC khi khai báo biến.

Định nghĩa cấu trúc

- Định nghĩa cấu trúc kiểu toàn cục
- Cú pháp

```
struct tên_cấu_trúc
{
        kiểu_1 tên_biến_1;
        kiểu_2 tên_biến_2;
        …
        kiểu_n tên_biến_n;
};
```

- Ví dụ:

```
struct CDAccountV1
{
        double balance;
        double interestRate;
        int term;
};
```

Khai báo biến cấu trúc

- Khi kiểu cấu trúc đã được định nghĩa, có thể dùng để khai báo biến cho kiểu cấu trúc này

  ví dụ *CDAccountV1 account;*
  - Giống như khai báo những kiểu cơ sở
  - Biến sau khi khai báo sẽ bao gồm các giá trị thành viên (member values)
- Truy cập đến những thành viên của cấu trúc sử dụng dấu .
  - vd: *account.balance, account.interestRate, account.term*
- Các cấu trúc khác nhau có thể có tên thành viên trùng nhau

Chương trình với cấu trúc (1/3)

Display 6.1 A Structure Definition

```
1  //Program to demonstrate the CDAccountV1 structure type.
2  #include <iostream>
3  using namespace std;

4  //Structure for a bank certificate of deposit:
5  struct CDAccountV1
6  {
7      double balance;
8      double interestRate;
9      int term;//months until maturity
10 };

11 void getData(CDAccountV1& theAccount);
12 //Postcondition: theAccount.balance, theAccount.interestRate, and
13 //theAccount.term have been given values that the user entered at the keyboar
```

An improved version of this structure will be given later in this chapter.

Chương trình với cấu trúc (2/3)

```
14  int main( )
15  {
16      CDAccountV1 account;
17      getData(account);

18      double rateFraction, interest;
19      rateFraction = account.interestRate/100.0;
20      interest = account.balance*(rateFraction*(account.term/12.0));
21      account.balance = account.balance + interest;

22      cout.setf(ios::fixed);
23      cout.setf(ios::showpoint);
24      cout.precision(2);
25      cout << "When your CD matures in "
26           << account.term << " months,\n"
27           << "it will have a balance of $"
28           << account.balance << endl;

29      return 0;
30  }
```

(continued)

Chương trình với cấu trúc (3/3)

Display 6.1 **A Structure Definition**

```
31  //Uses iostream:
32  void getData(CDAccountV1& theAccount)
33  {
34      cout << "Enter account balance: $";
35      cin >> theAccount.balance;
36      cout << "Enter account interest rate: ";
37      cin >> theAccount.interestRate;
38      cout << "Enter the number of months until maturity: ";
39      cin >> theAccount.term;
40  }
```

SAMPLE DIALOGUE

Enter account balance: **$100.00**
Enter account interest rate: **10.0**
Enter the number of months until maturity: **6**
When your CD matures in 6 months,
it will have a balance of $105.00

Lưu ý

- Khi định nghĩa cấu trúc phải kết thúc bằng dấu ; (semicolon)

```
struct WeatherData
{
        double temperature;
        double windVelocity;
}; ←YÊU CẦU dấu ;
```

Gán cấu trúc

- Giả sử khai báo 2 biến: *CDAccountV1 account1, account2;*
- Phép gán *account1 = account2;* là hợp lệ
  - Copy giá trị của mỗi biến thành viên từ *account2* sang *account1*

Sử dụng cấu trúc như đối số của hàm

- Truyền biến kiểu cấu trúc vào hàm giống như các kiểu cơ sở khác
  - Tham trị (pass-by-value)
  - Tham chiếu (pass-by-reference)
  - Hoặc kết hợp cả hai
- Có thể sử dụng kiểu cấu trúc như kiểu trả về của hàm

Khởi tạo biến kiểu cấu trúc

- Có thể khởi tạo biến kiểu cấu trúc khi khai báo

```
struct Date
{
        int month;
        int day;
        int year;
};
Date dueDate = {12, 31, 2003};
```

Bài tập

- Viết một chương trình khai báo một cấu trúc *Student* gồm các thông tin:
  - *Mã sinh viên: string*
  - *Tên sinh viên: string / char[20]*
  - *Lớp: string*
  - *Điểm trung bình: float*

Nhập giá trị cho N sinh viên (N < 10), hiển thị thông tin từng sinh viên và cho biết sinh viên nào có điểm trung bình lớn nhất.

2. KIỂU LỚP

(Class)

Mục tiêu bài học

- Định nghĩa lớp
- Hàm thành viên
- Thành viên *public* và *private*
- Hàm *accessor* và *mutator*
- Cấu trúc và lớp

Định nghĩa lớp (1/2)

- Tương tự như cấu trúc, NHƯNG
  - Lớp không chỉ có dữ liệu thành viên như cấu trúc,
  - Lớp còn bao gồm các HÀM thành viên để thao tác trên dữ liệu
- Khái niệm LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (object-oriented programming - OOP)
  - Nhìn dưới góc độ đối tượng (object)
  - Object: gồm DỮ LIỆU + XỬ LÝ
  - Trong C++, các biến của một kiểu lớp là những đối tượng

Định nghĩa lớp (2/2)

- Định nghĩa tương tự như cấu trúc

```
class DayOfYear                ← tên của kiểu lớp
{
    public:
        void output();         ← tên hàm thành viên!
        int month;
        int day;
};
```

- Lớp là một kiểu đầy đủ (full-fledged type), giống như các kiểu dữ liệu cơ sở như *int*, *double*, *float*, *etc*.
- Tham số dùng cho một kiểu lớp có thể là:
  - Tham trị (pass-by-value)
  - Tham biến (pass-by-reference)

Khai báo đối tượng

- Khai báo giống như các biến của các kiểu cơ sở
  - *đối tượng là một biến của kiểu lớp*
- Ví dụ: *DayOfYear today, birthday;*
  - Khai báo 2 đối tượng *today*, *birthday* của lớp *DayOfYear*
- Một đối tượng bao gồm:
  - Dữ liệu. vd *month, day*
  - Thao tác (hàm thành viên). vd *output()*

Truy cập thành viên của lớp

- Truy cập vào các thành viên của lớp tương tự như cấu trúc: sử dụng dấu .
- Ví dụ: *today.month, today.day*
  - Gọi hàm thành viên: *today.output()*

Hàm thành viên của lớp

- Phải khai báo và cài đặt (implement) các hàm thành viên của lớp
- Giống như các hàm thông thường, hàm thành viên của lớp có thể được định nghĩa sau *main()*
  - Phải chỉ định tên lớp đi kèm
  - Ví dụ: *void DayOfYear::output()* {…}
  - Toán tử :: gọi là toán tử phân giải phạm vi (scope resolution operator)

Chương trình với lớp (1/4)

Display 6.3 **Class with a Member Function**

```
1  //Program to demonstrate a very simple example of a class.
2  //A better version of the class DayOfYear will be given in Display 6.4.
3  #include <iostream>
4  using namespace std;

5  class DayOfYear
6  {
7  public:
8      void output( );
9      int month;
10     int day;
11 };

12 int main( )
13 {
14     DayOfYear today, birthday;
15     cout << "Enter today's date:\n";
16     cout << "Enter month as a number: ";
17     cin >> today.month;
18     cout << "Enter the day of the month: ";
19     cin >> today.day;
20     cout << "Enter your birthday:\n";
21     cout << "Enter month as a number: ";
22     cin >> birthday.month;
23     cout << "Enter the day of the month: ";
24     cin >> birthday.day;
```

Normally, member variables are `private` *and not* `public`*, as in this example. This is discussed a bit later in this chapter.*

Member function declaration

(continued)

Chương trình với lớp (2/4)

Display 6.3 Class with a Member Function

```
25      cout << "Today's date is ";
26      today.output( );
27      cout << endl;
28      cout << "Your birthday is ";
29      birthday.output( );
30      cout << endl;

31      if (today.month == birthday.month && today.day == birthday.day)
32          cout << "Happy Birthday!\n";
33      else
34          cout << "Happy Unbirthday!\n";
35      return 0;
36  }
37  //Uses iostream:
38  void DayOfYear::output( )
39  {
40      switch (month)
41      {
42          case 1:
43              cout << "January "; break;
44          case 2:
45              cout << "February "; break;
46          case 3:
47              cout << "March "; break;
48          case 4:
49              cout << "April "; break;
```

Calls to the member function **output**

Member function definition

Chương trình với lớp (3/4)

```
50          case 5:
51              cout << "May "; break;
52          case 6:
53              cout << "June "; break;
54          case 7:
55              cout << "July "; break;
56          case 8:
57              cout << "August "; break;
58          case 9:
59              cout << "September "; break;
60          case 10:
61              cout << "October "; break;
62          case 11:
63              cout << "November "; break;
64          case 12:
65              cout << "December "; break;
66          default:
67              cout << "Error in DayOfYear::output. Contact software vendor.";
68      }
69
70      cout << day;
71  }
```

Chương trình với lớp (3/4)

Display 6.3 **Class with a Member Function**

SAMPLE DIALOGUE

```
Enter today's date:
Enter month as a number: 10
Enter the day of the month: 15
Enter your birthday:
Enter month as a number: 2
Enter the day of the month: 21
Today's date is October 15
Your birthday is February 21
Happy Unbirthday!
```

Toán tử . Và ::

- Đều được sử dụng để chỉ định thành viên của một thực thể nào đó
- Toán tử .
  - Chỉ định thành viên của một ĐỐI TƯỢNG XÁC ĐỊNH
- Toán tử ::
  - Chỉ định một hàm thuộc về LỚP nào

Đóng gói (Encapsulation)

- Đóng gói là: *Tập hợp nhiều thứ cùng nhau vào một thực thể đơn*
- Một kiểu dữ liệu bất kỳ bao gồm:
  - Dữ liệu (khoảng dữ liệu)
  - Thao tác trên dữ liệu đó (operations)
- Ví dụ: kiểu dữ liệu *int* có
  - Dữ liệu: +-32,767
  - Thao tác: +,-,*,/,%,logical, …
- Tương tự với lớp NHƯNG chúng ta gộp dữ liệu và những thao tác trên dữ liệu đó vào trong một lớp

Kiểu dữ liệu trừu tượng (Abstract)

- Thế nào là *Trừu tượng*: lập trình viên không biết chi tiết về kiểu dữ liệu đó
- Viết tắt là ADT:
  - Tập hợp của những giá trị dữ liệu cùng nhau với một tập những thao tác cơ bản được định nghĩa cho các giá trị này
- ADT thường độc lập ngôn ngữ (language-independent)
  - Cài đặt ADT trong C++ với khái niệm lớp (class)

Những nguyên lý của OOP

- Che dấu thông tin
  - Người dùng không biết / không cần biết chi tiết về cách thức hoạt động / cài đặt của các thao tác (hàm)
- Dữ liệu trừu tượng (Data Abstraction)
  - Người dùng không biết chi biết dữ liệu được xử lý như thế nào bên trong ADT/lớp
- Đóng gói (encapsulation)
  - Tập hợp dữ liệu và những thao tác trên dữ liệu cùng nhau, nhưng che dấu chi tiết về chúng
- Mục đích?

Thành viên **public** và **private**

- Dữ liệu trong một lớp hầu như luôn được thiết kế là ***private*** khi định nghĩa
  - Tuân theo nguyên tắc của OOP
  - Che dấu dữ liệu với người dùng
  - Cho phép xử lý chỉ thông qua các hàm thành viên / thao tác
- Người dùng chỉ có thể truy cập được (user-accessible) các hàm hoặc dữ liệu public
- Bên ngoài định nghĩa của lớp, không thể thay đổi (hoặc thậm chí truy cập) được dữ liệu private

Ví dụ về public và private

```
class DayOfYear
{
public:
        void input();
        void output();
private:
        int month;
        int day;
};
```

- Dữ liệu là *private* => không thể truy cập được bởi người dùng
- Khai báo đối tượng: *DayOfYear today;*
- Đối tượng *today* CHỈ có thể truy cập được các thành viên public
  - *cin >> today.month; // KHÔNG CHO PHÉP!*
  - *cout << today.day; // KHÔNG CHO PHÉP!*
  - Những lời gọi sau là hợp lý
    - *today.input();*
    - *today.output();*

Hàm **accessor** và **mutator**

- Đối tượng cần "làm một cái gì đó" với dữ liệu của nó
- Gọi là hàm thành viên accessor nếu:
  - Cho phép đối tượng ĐỌC dữ liệu
  - Cũng được gọi là hàm thành viên GET
  - Đơn giản là lấy dữ liệu thành viên
- Gọi là hàm thành viên mutator nếu:
  - Cho phép đối tượng THAY ĐỔI dữ liệu

Giao diện tách rời và cài đặt (Separate Interface & Implementation)

- Người dùng của lớp không cần nhìn thấy chi tiết của việc lớp đó được cài đặt như thế nào
  - Nguyên tác của OOP: đóng gói
- Người dùng chỉ cần "những quy tắc (rules)"
  - Được gọi là "giao diện (interface)" cho lớp đó
  - Trong C++, giao diện bao gồm những hàm thành viên public và những chú thích đi kèm
- Cài đặt của lớp bị che đi
  - Định nghĩa hàm thành viên ở một nơi nào đó
  - Người dùng không cần nhìn thấy chúng

So sánh cấu trúc và lớp

- Cấu trúc
  - Không có hàm thành viên
  - Tất cả dữ liệu thành viên đều là public
- Lớp
  - Có hàm thành viên
  - Dữ liệu thành viên thường (mặc định) là private
  - Giao diện các hàm thành viên là public

Suy nghĩ về đối tượng

- Trọng tâm của việc lập trình thay đổi
  - Giai đoạn trước: thuật toán (algorithms) là trung tâm
  - Với OOP: tập trung vào dữ liệu
- Thiết kế giải pháp phần mềm:
  - Định nghĩa nhiều đối tượng và cách thức giao tiếp giữa chúng

Bài tập

- Viết một chương trình C++ khai báo lớp Sinh_Viên bao gồm:
  - Biến Thành viên (Dữ liệu):
    - *ID: int*
    - *Tên_Sinh_Viên : char[50]*
    - *Điểm_Kiểm_tra_Giữa_Kỳ: float*
    - *Điểm_Kiểm_Tra_Cuối_Kỳ: float*
  - Hàm thành viên:
    - **input**(): Nhập thông tin sinh viên
    - **output**(): In ra màn hình thông tin sinh viên
    - **editDiemGiuaKy**(): Sửa điểm giữa kỳ của sinh viên

TÓM TẮT

- Cấu trúc là tập hợp những kiểu dữ liệu khác nhau
- Lớp được dùng để kết hợp dữ liệu và hàm thành một đơn vị đơn, gọi là đối tượng (object)
- Biến thành viên và hàm thành viên
  - Có thể public: truy cập được ở bên ngoài lớp
  - Có thể private: chỉ được phép truy cập bên trong lớp (trong định nghĩa của các hàm thành viên)
- Cả lớp và cấu trúc đều có thể là tham số cho các hàm thông thường khác
- Định nghĩa lớp trong C++ nên chia thành 2 phần:
  - Giao diện (interface): phần mà người dùng cần
  - Cài đặt (implementation): chi tiết của việc lớp làm việc như thế nào

3. Hàm tạo & Hàm Hủy

VÀ CÁC CÔNG CỤ KHÁC

MỤC TIÊU BÀI HỌC

- Hàm tạo (Constructor)
  - Định nghĩa hàm tạo
  - Gọi hàm tạo
- Hàm hủy (Destructor)
- Các công cụ khác
  - Hàm inline
  - Thành viên tĩnh (static members)
- Lớp Vector

Hàm tạo (Constructor)

- Khởi tạo cho đối tượng của một lớp
  - Khởi tạo một vài hoặc tất cả các biến thành viên
  - Kèm theo một vài hoạt động khác
- Một loại đặc biệt của hàm thành viên
  - Tự động được gọi khi khai báo đối tượng
- Một công cụ hữu ích: nguyên tắc quan trọng của OOP
- Hàm tạo được định nghĩa giống như các hàm thành viên khác, NGOẠI TRỪ:
  - Tên hàm tạo trùng với tên lớp
  - Không trả lại giá trị, thậm chí là void
- Hàm tạo được đặt ở trong phần public

Ví dụ hàm tạo

```
class DayOfYear
{
public:
    // constructor initializes month & day
    DayOfYear (int monthValue, int dayValue);
    void input();
    void output();
    …
private:
    int month;
    int day;
}
```

Gọi hàm tạo

- Khai báo đối tượng: *DayOfYear date1(7,4), date2(5,5);*
- Khi đối tượng *date1*, *date2* được tạo
  - Hàm tạo được gọi
  - Các giá trị được truyền vào như đối số của hàm tạo
  - Các biến thành viên *month, day* được khởi tạo

  date1.month → 7, date2.month → 5
  date1.day → 4, date2.day → 5
- CHÚ Ý: không thể gọi hàm tạo một cách tường minh như các hàm thành viên khác

```
date1.DayOfYear(7, 4);      // KHÔNG HỢP LỆ!
date2.DayOfYear(5, 5);      // KHÔNG HỢP LỆ!
```

CÀI ĐẶT HÀM TẠO

- Cài đặt hàm tạo giống như các hàm thành viên khác

```
DayOfYear::DayOfYear(int monthValue, int dayValue)
{
        month = monthValue;
        day = dayValue;
}
```

- Chú ý 2 tên giống nhau xung quanh toán tử ::
- Không có kiểu trả về
- Một cách cài đặt hàm tạo khác tương đương:

```
DayOfYear::DayOfYear(int monthValue, int dayValue)
:month(monthValue), day(dayValue) {}
```

  - Với phần thân hàm để trống

Mục đích khác nữa của hàm tạo

- Hàm tạo không chỉ dùng để khởi tạo dữ liệu
- Hàm tạo có thể được dùng vào nhiều mục đích khác
  - Kiểm tra tính hợp lệ của dữ liệu
  - Thực thi các hoạt động khác liên quan

Hàm tạo nạp chồng

- Hàm tạo có thể được nạp chồng giống như các hàm thông thường khác
- Nhớ lại: chữ ký (signature) để nhận diện một hàm bao gồm: <*Tên hàm + danh sách tham số*>
- Có thể cung cấp hàm tạo cho tất cả trường hợp có thể với danh sách đối số khác nhau (nhưng cần thiết)

Chương trình với hàm tạo (1/3)

Display 7.1 Class with Constructors

```
1   #include <iostream>
2   #include <cstdlib> //for exit
3   using namespace std;

4   class DayOfYear
5   {
6   public:
7       DayOfYear(int monthValue, int dayValue);
8       //Initializes the month and day to arguments.

9       DayOfYear(int monthValue);
10      //Initializes the date to the first of the given month.

11      DayOfYear( );
12      //Initializes the date to January 1.

13      void input( );
14      void output( );
15      int getMonthNumber( );
16      //Returns 1 for January, 2 for February, etc.
```

This definition of `DayOfYear` *is an improved version of the class* `DayOfYear` *given in Display 6.4.*

default constructor

Chương trình với hàm tạo (2/3)

```
17      int getDay( );
18  private:
19      int month;
20      int day;
21      void testDate( );
22  };

23  int main( )
24  {
25      DayOfYear date1(2, 21), date2(5), date3;
26      cout << "Initialized dates:\n";
27      date1.output( ); cout << endl;
28      date2.output( ); cout << endl;
29      date3.output( ); cout << endl;

30      date1 = DayOfYear(10, 31);
31      cout << "date1 reset to the following:\n";
32      date1.output( ); cout << endl;
33      return 0;
34  }
35
36  DayOfYear::DayOfYear(int monthValue, int dayValue)
37                              : month(monthValue), day(dayValue)
38  {
39      testDate( );
40  }
```

This causes a call to the default constructor. Notice that there are no parentheses.

an explicit call to the constructor `DayOfYear::DayOfYear`

Chương trình với hàm tạo (3/3)

Display 7.1 Class with Constructors

```
41  DayOfYear::DayOfYear(int monthValue) : month(monthValue), day(1)
42  {
43      testDate( );
44  }

45  DayOfYear::DayOfYear( ) : month(1), day(1)
46  {/*Body intentionally empty.*/}

47  //uses iostream and cstdlib:
48  void DayOfYear::testDate( )
49  {
50      if ((month < 1) || (month > 12))
51      {
52          cout << "Illegal month value!\n";
53          exit(1);
54      }
55      if ((day < 1) || (day > 31))
56      {
57          cout << "Illegal day value!\n";
58          exit(1);
59      }
60  }
```

<Definitions of the other member functions are the same as in Display 6.4.>

SAMPLE DIALOGUE

```
Initialized dates:
February 21
May 1
January 1
date1 reset to the following:
October 31
```

Hàm tạo không có đối số

- Hàm chuẩn (thông thường) khai báo không có đối số, khi gọi có dấu (). Ví dụ: *callMyFunction();*
- Khi đối tượng khai báo mà không có hàm tạo, hoặc có hàm tạo nhưng không có tham số
  - *DayOfYear date1;* *// ĐÚNG!*
  - *DayOfYear date();* *// SAI!*

Hàm tạo mặc định (default)

- Là: hàm tạo không có đối số
- Nên định nghĩa một hàm tạo mặc định
- **Hàm tạo mặc định có được tự động sinh ra?**
  - Vừa đúng vừa sai
  - Nếu không có bất kỳ hàm tạo nào được định nghĩa -> Đúng
  - Nếu có ít nhất 1 hàm tạo được định nghĩa -> Sai
- Nếu không có hàm tạo mặc định:
  - Không thể khai báo đối tượng

Gọi tường minh hàm tạo

- Bạn cũng có thể gọi lại hàm tạo sau khi đã khai báo đối tượng
  - Nhớ lại: hàm tạo được tự động gọi khi khai báo đối tượng
  - Lời gọi này trả lại một đối tượng vô danh (anonymous object)
- Ví dụ: *DayOfYear holiday(7, 4);*
  - Khai báo đối tượng *holiday* sẽ gọi hàm tạo của lớp *DayOfYear*
  - Bây giờ khởi tạo lại (re-initialize)

    holiday = DayOfYear(5,5);
    - Gọi hàm tạo một cách tường minh (explicit call)
    - Trả về một đối tượng mới vô danh (anonymous object)
    - Gán ngược lại cho đối tượng *holiday*

Kiểu biến thành viên của lớp (1/5)

- Biến thành viên của lớp có thể là bất kỳ kiểu nào
  - Bao gồm đối tượng của những lớp khác !

Display 7.3 A Class Member Variable

```
1  #include <iostream>
2  #include<cstdlib>
3  using namespace std;

4  class DayOfYear
5  {
6  public:
7      DayOfYear(int monthValue, int dayValue);
8      DayOfYear(int monthValue);
9      DayOfYear( );
10     void input( );
11     void output( );
12     int getMonthNumber( );
13     int getDay( );
14 private:
15     int month;
16     int day;
17     void testDate( );
18 };
```

The class `DayOfYear` *is the same as in Display 7.1, but we have repeated all the details you need for this discussion.*

Kiểu biến thành viên của lớp (2/5)

```
19  class Holiday
20  {
21  public:
22      Holiday( );//Initializes to January 1 with no parking enforcement
23      Holiday(int month, int day, bool theEnforcement);
24      void output( );
25  private:
26      DayOfYear date;
27      bool parkingEnforcement;//true if enforced
28  };

29  int main( )
30  {
31      Holiday h(2, 14, true);
32      cout << "Testing the class Holiday.\n";
33      h.output( );

34      return 0;
35  }
36
37  Holiday::Holiday( ) : date(1, 1), parkingEnforcement(false)
38  {/*Intentionally empty*/}

39  Holiday::Holiday(int month, int day, bool theEnforcement)
40                       : date(month, day), parkingEnforcement(theEnforcement)
41  {/*Intentionally empty*/}
```

member variable of a class type

Invocations of constructors from the class `DayOfYear`.

(continued)

Kiểu biến thành viên của lớp (3/5)

Display 7.3 **A Class Member Variable**

```
42  void Holiday::output( )
43  {
44      date.output( );
45      cout << endl;
46      if (parkingEnforcement)
47          cout << "Parking laws will be enforced.\n";
48      else
49          cout << "Parking laws will not be enforced.\n";
50  }

51  DayOfYear::DayOfYear(int monthValue, int dayValue)
52                              : month(monthValue), day(dayValue)
53  {
54      testDate( );
55  }
```

Kiểu biến thành viên của lớp (4/5)

```
56  //uses iostream and cstdlib:
57  void DayOfYear::testDate( )
58  {
59      if ((month < 1) || (month > 12))
60      {
61          cout << "Illegal month value!\n";
62          exit(1);
63      }
64      if ((day < 1) || (day > 31))
65      {
66          cout << "Illegal day value!\n";
67          exit(1);
68      }
69  }
70
71  //Uses iostream:
72  void DayOfYear::output( )
73  {
74      switch (month)
75      {
76          case 1:
77              cout << "January "; break;
78          case 2:
79              cout << "February "; break;
80          case 3:
81              cout << "March "; break;
              .
              .
              .
```

The omitted lines are in Display 6.3, but they are obvious enough that you should not have to look there.

Kiểu biến thành viên của lớp (5/5)

Display 7.3 **A Class Member Variable**

```
82            case 11:
83                cout << "November "; break;
84            case 12:
85                cout << "December "; break;
86            default:
87                cout << "Error in DayOfYear::output. Contact software vendor.";
88        }

89        cout << day;
90    }
```

SAMPLE DIALOGUE

```
Testing the class Holiday.
February 14
Parking laws will be enforced.
```

Hàm hủy (Destructor)

- Là một hàm thành viên của lớp có chức năng ngược với Hàm tạo
- Được gọi trước khi giải phóng (xoá bỏ) một đối tượng
- Cú pháp:**~Tên_Lớp()**
- Ví dụ:

```
class Da_Thuc
{
    private:
        int n;  // bậc đa thức
        double *a; // trỏ tới vùng nhớ chứa các hệ số đa thức
        ...
    public:
        ~Da_Thuc()
        {
                n=0;
                delete a;  a= NULL;
        }
        ...
};
```

Bài tập cho hàm tạo

- Sử dụng Hàm tạo và Hàm hủy cho chương trình C++ khai báo lớp **Sinh_Viên** bao gồm:
  - Dữ liệu:
    - *ID,*
    - *Tên_Sinh_Viên,*
    - *Điểm_Kiểm_tra_Giữa_Kỳ,*
    - *Điểm_Kiểm_Tra_Cuối_Kỳ*
  - Các thao tác như:
    - Nhập thông tin sinh viên
    - In ra màn hình thông tin sinh viên
    - Sửa thông tin sinh viên

Các phương pháp truyền tham số

- Hiệu quả của truyền tham số
  - Tham trị (call-by-value)
    - Một bản sao của tham số được tạo ra
  - Tham biến (call-by-reference)
    - Dùng địa chỉ / tham chiếu (reference/placeholder) của chính tham số đó
  - Hai phương pháp này khác nhau không đáng kể với những kiểu đơn giản
  - NHƯNG với kiểu lớp, truyền tham chiếu có lợi ích đáng kể, đặc biệt với dữ liệu lớn
- Để đảm bảo tham số không thay đổi, sử dụng từ khóa *const*.

Hàm nội tuyến (Inline functions)

- Trình biên dịch đặt một bản sao code của thân hàm nội tuyến tại mỗi vị trí mà hàm đó được gọi tại thời gian biên dịch (compile time).
- Thường được dùng để loại bỏ thời gian quá dụng (*overhead*) xảy ra khi gọi một hàm, cho phép tăng tốc độ thực hiện chương trình nhưng lại chiếm không gian bộ nhớ nhiều hơn
- Chỉ sử dụng cho những hàm nhỏ và được dùng thường xuyên. Ví dụ: *max(a,b)*
- Với những hàm không phải hàm thành viên của lớp
  - Sử dụng từ khóa *inline* trong khai báo hàm và trước tên hàm
- Với những hàm là hàm thành viên của lớp
  - Đặt cài đặt (code) của hàm BÊN TRONG định nghĩa lớp -> tự động inline

Thành viên tĩnh (Static members)

- Biến thành viên tĩnh
  - Mọi đối tượng của lớp đều chia sẻ duy nhất MỘT bản sao của biến thành viên đó
  - Khi một đối tượng thay đổi giá trị của biến đó -> mọi đối tượng khác đều thấy sự thay đổi
- Hữu ích cho việc truy vết (tracking)
  - Một hàm thành viên có được gọi thường xuyên không?
  - Đối tượng tồn tại bao nhiêu lần tại một thời điểm?
- Khai báo bằng cách đặt từ khóa *static* trước kiểu

Ví dụ hàm nội tuyến (inline)

```
inline double cube(double side)
{
    return side * side * side
}

void main()
{
    double dSideValue=4;
    cout << cube(dSideValue) << endl;
}
```

Hàm tĩnh

- Hàm thành viên cũng có thể là tĩnh (static)
  - Nếu không cần thiết truy cập đến dữ liệu của đối tượng
  - Và vẫn phải là thành viên của lớp
- Hàm tĩnh chỉ có thể dùng dữ liệu tĩnh, gọi hàm tĩnh khác
- Có thể được gọi bên ngoài lớp chứa hàm đó
  - Từ những đối tượng không thuộc lớp (non-class), vd. *Server::getTurn();*
  - Hoặc thông qua đối tượng thuộc lớp đó,
    - Phương thức chuẩn: *myObject.getTurn();*

Chương trình cho thành viên tĩnh (1/4)

Display 7.6 Static Members

```
1  #include <iostream>
2  using namespace std;

3  class Server
4  {
5  public:
6      Server(char letterName);
7      static int getTurn( );
8      void serveOne( );
9      static bool stillOpen( );
10 private:
11     static int turn;
12     static int lastServed;
13     static bool nowOpen;
14     char name;
15 };

16 int Server:: turn = 0;
17 int Server:: lastServed = 0;
18 bool Server::nowOpen = true;
```

Chương trình cho thành viên tĩnh (2/4)

```
19  int main( )
20  {
21      Server s1('A'), s2('B');
22      int number, count;
23      do
24      {
25          cout << "How many in your group? ";
26          cin >> number;
27          cout << "Your turns are: ";
28          for (count = 0; count < number; count++)
29              cout << Server::getTurn( ) << ' ';
30          cout << endl;
31          s1.serveOne( );
32          s2.serveOne( );
33      } while (Server::stillOpen( ));
34      cout << "Now closing service.\n";
35      return 0;
36  }
37
38
```

Chương trình cho thành viên tĩnh (3/4)

Display 7.6 Static Members

```
39  Server::Server(char letterName) : name(letterName)
40  {/*Intentionally empty*/}

41  int Server::getTurn( )
42  {
43      turn++;
44      return turn;
45  }
46  bool Server::stillOpen( )
47  {
48      return nowOpen;
49  }

50  void Server::serveOne( )
51  {
52      if (nowOpen && lastServed < turn)
53      {
54          lastServed++;
55          cout << "Server " << name
56              << " now serving " << lastServed << endl;
57       }
```

Since `getTurn` *is static, only static members can be referenced in here.*

Chương trình cho thành viên tĩnh (4/4)

```
58        if (lastServed >= turn) //Everyone served
59            nowOpen = false;
60    }
```

SAMPLE DIALOGUE

How many in your group? **3**
Your turns are: 1 2 3
Server A now serving 1
Server B now serving 2
How many in your group? **2**
Your turns are: 4 5
Server A now serving 3
Server B now serving 4
How many in your group? **0**
Your turns are:
Server A now serving 5
Now closing service.

Vector

- Nhớ lại: mảng (array) có kích thước cố định
- Vector là "*mảng nhưng kích thước có thể thay đổi*" trong lúc chạy chương trình
  - Có kiểu cơ sở và dùng để lưu một tập các giá trị kiểu cơ sở
  - Là một kiểu lớp
- Sử dụng Thư Viện Mẫu Chuẩn (Standard Template Library)
- Cú pháp khai báo: *vector<Kiểu> tên_vector;*
- Ví dụ: *vector<int> v;*
  - Tạo ra một lớp mới cho vector với kiểu *int*
- Truy xuất các phần tử giống như mảng, thông qua index
- Nhưng khi thêm phần tử: phải gọi hàm thành viên *push_back()*
- Hàm thành viên *size()*: trả lại số lượng phần tử hiện tại của vector

Chương trình cho vector (1/2)

Display 7.7 Using a Vector

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;

4  int main( )
5  {
6      vector<int> v;
7      cout << "Enter a list of positive numbers.\n"
8           << "Place a negative number at the end.\n";

9      int next;
10     cin >> next;
11     while (next > 0)
12     {
13         v.push_back(next);
14         cout << next << " added. ";
15         cout << "v.size( ) = " << v.size( ) << endl;
16         cin >> next;
17     }
```

Chương trình cho vector (2/2)

```
18        cout << "You entered:\n";
19        for (unsigned int i = 0; i < v.size( ); i++)
20            cout << v[i] << " ";
21        cout << endl;

22        return 0;
23    }
```

SAMPLE DIALOGUE

Enter a list of positive numbers.
Place a negative number at the end.
2 4 6 8 -1
2 added. v.size = 1
4 added. v.size = 2
6 added. v.size = 3
8 added. v.size = 4
You entered:
2 4 6 8

Tóm tắt hàm tạo, hàm hủy

- Hàm tạo: tự động khởi tạo dữ liệu của lớp
  - Được gọi khi đối tượng của lớp được khai báo
  - Hàm tạo có cùng tên với lớp
- Hàm tạo mặc định không có tham số
  - Nên luôn được định nghĩa
- Biến thành viên của lớp
  - Có thể là đối tượng của một lớp khác
- Hằng số tham chiếu hiệu quả hơn tham trị
- Sử dụng inline với các hàm nhỏ để tăng hiệu suất
- Biến thành viên tĩnh: được chia sẻ bởi tất cả các đối tượng của lớp
- Lớp vector: là "mảng nhưng kích thước có thể thay đổi"

Tham khảo

- **Giáo trình chính: W. Savitch, *Absolute C++*, Addison Wesley, 2002**
- Tham khảo:
  - A. Ford and T. Teorey, *Practical Debugging in C++*, Prentice Hall, 2002
  - Nguyễn Thanh Thủy, *Kĩ thuật lập trình C++*, NXB Khoa học và Kĩ Thuật, 2006