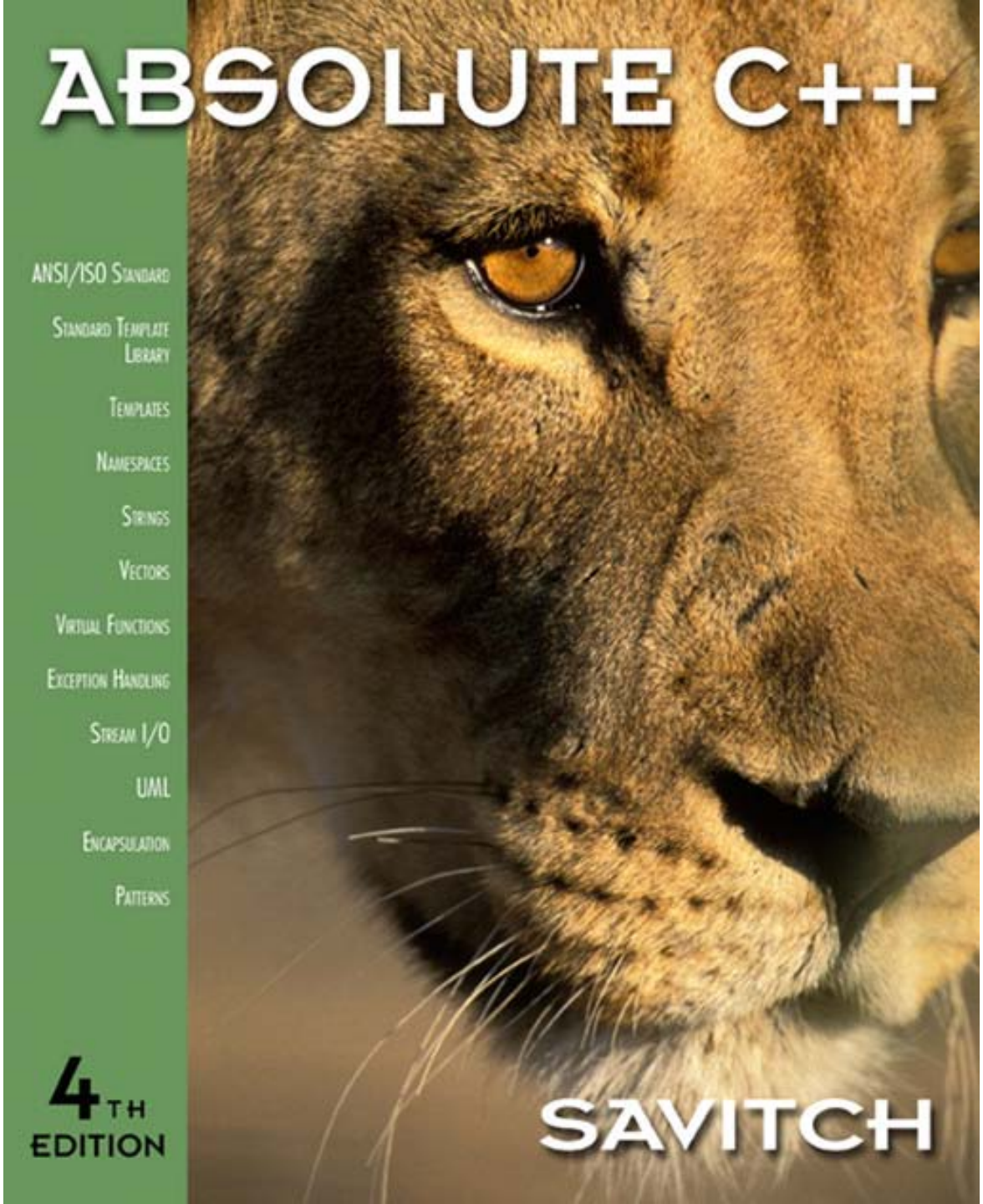
Bài 7: Hàm kiến tạo và các công cụ khác

Giảng viên: Hoàng Thị Điệp
Khoa Công nghệ Thông tin – ĐH Công Nghệ



Chapter 7

Constructors and Other Tools

PEARSON Addison Wesley

Mục tiêu bài học

- Hàm kiến tạo
  - Định nghĩa
  - Cách gọi
- Một số công cụ
  - Từ khóa const cho tham số
  - Hàm inline
  - Thành viên dữ liệu static
- vector
  - Giới thiệu lớp vector

Hàm kiến tạo

- Khởi tạo đối tượng
  - Khởi tạo một vài hay tất cả các biến thành viên
  - Có thể thực hiện thêm các công việc khác
- Một kiểu hàm thành viên đặc biệt
  - Được gọi tự động khi một đối tượng được khai báo
- Công cụ rất hữu ích
  - Nguyên lý chính của LTHĐT

Định nghĩa hàm kiến tạo

- Hàm kiến tạo được định nghĩa giống như các hàm thành viên khác
  - Ngoại trừ:
    1. Nó phải có tên giống hệt tên lớp
    2. Không thể trả về một giá trị; kể cả void!

Ví dụ định nghĩa hàm kiến tạo

- Định nghĩa lớp với hàm kiến tạo:

```
class DayOfYear
{
public:
    DayOfYear(int monthValue, int dayValue);
            //Hàm kiến tạo khởi tạo month & day
    void input();
    void output();
private:
    int month;
    int day;
}
```

Lưu ý về hàm kiến tạo

- Tên của hàm kiến tạo: DayOfYear
  - Trùng tên lớp
- Trong khai báo hàm kiến tạo không chỉ định kiểu trả về
  - Kể cả là kiểu void
- Hàm kiến tạo nằm trong vùng public
  - Nó được gọi mỗi khi một đối tượng được khai báo
  - Nếu private, sẽ không thể khai báo được đối tượng.

Gọi tới hàm kiến tạo

- Khai báo đối tượng:

```
DayOfYear    date1(7, 4),
             date2(5, 5);
```

- Các đối tượng sẽ được tạo ra ở đây.
  - Hàm kiến tạo được gọi
  - Các giá trị trong cặp ngoặc tròn được truyền làm số của hàm kiến tạo
  - Các biến thành viên month, day được khởi tạo:
    date1.month → 7 date2.month → 5
    date1.day → 4 date2.day → 5

Thử lời gọi tương đương

- Xét:

```
DayOfYear date1, date2
date1.DayOfYear(7, 4);     // không hợp lệ!
date2.DayOfYear(5, 5);     // không hợp lệ!
```

- Có vẻ là ổn…
  - KHÔNG thể gọi tới hàm kiến tạo như cách bạn vẫn gọi tới các hàm thành viên!

Mã nguồn hàm kiến tạo

- Định nghĩa hàm kiến tạo cũng giống định nghĩa các hàm thành viên khác:

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

- Chú ý 2 tên giống nhau ở 2 bên ::
  - Nhận diện rõ ràng một hàm kiến tạo
- Chú ý không có kiểu trả về
  - Nhất quán với định nghĩa lớp

Một cách định nghĩa khác

- Định nghĩa lúc trước tương đương với:

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

- Dòng trỏ bởi mũi tên gọi là “vùng khởi tạo”
- Thân của hàm này rỗng
- Cách định nghĩa này được khuyến khích dùng hơn

Mục đích gia tăng của hàm kiến tạo

- Không chỉ nhằm khởi tạo dữ liệu
- Thân hàm không nhất thiết phải rỗng
- Mà còn nhằm kiểm tra tính hợp lệ của dữ liệu
  - Đảm bảo chỉ những dữ liệu hợp lý mới được gán cho biến thành viên của lớp
  - Nguyên lý quan trọng LTHĐT

Nạp chồng hàm kiến tạo

- Có thể nạp chồng hàm kiến tạo như làm với các hàm khác
- Nhắc lại: chữ kí của hàm bao gồm
  - tên hàm
  - danh sách tham số
- Nên cung cấp hàm kiến tạo cho tất cả các danh sách đối số có thể
  - Cụ thể là “bao nhiêu” đối số

Ví dụ lớp có hàm kiến tạo: **Display 7.1** Lớp có hàm kiến 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

Ví dụ lớp có hàm kiến tạo:
Display 7.1 Lớp có hàm kiến 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`

Ví dụ lớp có hàm kiến tạo: **Display 7.1** Lớp có hàm kiến 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 kiến tạo không đối số

- Có thể gây nhầm lẫn
- Một hàm không đối số chuẩn:
  - Được gọi với cú pháp: callMyFunction();
    - Cặp ngoặc tròn rỗng bên trong
- Ứng với lệnh khai báo đối tượng mà chỉ định giá trị khởi tạo:
  - DayOfYear date1; // Gọi hàm kiến tạo!
  - DayOfYear date(); // Không gọi hàm kiến tạo!
    - Điều gì thực sự diễn ra?
    - Trình biên dịch coi đây là một khai báo/nguyên mẫu hàm!
    - Hãy nhìn kĩ cấu trúc của dòng lệnh này!

Gọi tường minh tới hàm kiến tạo

- Có thể gọi tới hàm kiến tạo một lần nữa
  - Sau khi khai báo đối tượng
    - Nhắc lại: hàm kiến tạo được gọi tự động khi được khai báo
  - Có thể gọi thông qua tên đối tượng; lời gọi chuẩn tới hàm thành viên
- Cho ta một cách tiện lợi để lập giá trị các biến thành viên
- Cách này khá khác cách gọi chuẩn tới hàm thành viên

Ví dụ gọi tường minh tới hàm kiến tạo

- Lời gọi như vậy sẽ trả về "một đối tượng vô danh"
  - sau đó có thể dùng nó trong phép gán
  - **Hành động**:

    DayOfYear holiday(7, 4);
    - Hàm kiến tạo được gọi khi khai báo đối tượng
    - Bây giờ ta sẽ "khởi tạo lại" :

    holiday = DayOfYear(5, 5);
      - Gọi tường minh tới hàm kiến tạo
      - Trả về một "đối tượng vô danh" mới
      - Gán nó cho đối tượng hiện thời

Hàm kiến tạo mặc định

- Được định nghĩa là: hàm kiến tạo không đối số
- Nên luôn có một hàm như vậy
- Sinh tự động?
  - Có và Không
  - Nếu không có bất cứ hàm kiến tạo nào được định nghĩa → Có
  - Nếu có hàm kiến tạo nào đó đã được định nghĩa → Không
- Nếu không có hàm kiến tạo mặc định:
  - Không thể khai báo: MyClass myObject;
    - Không có giá trị khởi tạo đi kèm

Biến thành viên có kiểu định nghĩa bởi lớp

- Biến thành viên có thể thuộc bất cứ kiểu nào
  - Kể cả là đối tượng của lớp khác!
  - Kiểu quan hệ giữa các lớp
    - Nguyên lý quan trọng của LTHĐT
- Cần kí hiệu đặc biệt cho hàm kiến tạo
  - Để chúng có thể gọi "ngược" tới hàm kiến tạo của đối tượng thành viên

Ví dụ biến thành viên có kiểu class

Display 7.3 Biến thành viên có kiểu class (1/5)

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.*

Ví dụ biến thành viên có kiểu class

Display 7.3 Biến thành viên có kiểu class (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)

Ví dụ biến thành viên có kiểu class
Display 7.3 Biến thành viên có kiểu class (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  }
```

Ví dụ biến thành viên có kiểu class
Display 7.3 Biến thành viên có kiểu class (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.

Ví dụ biến thành viên có kiểu class

Display 7.3 Biến thành viên có kiểu class (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.
```

Cách truyền tham số

- Hiệu quả của việc truyền tham số
  - Truyền giá trị
    - Cần sao đối số → Sinh "chi phí"
  - Truyền tham chiếu
    - Chỗ đặt trước cho đối số thực sự
    - Là cách hiệu quả nhất
  - Khác biệt này là không đáng kể với kiểu dữ liệu đơn
  - Với kiểu định nghĩa bởi lớp → lợi thế rõ ràng
- Truyền tham chiếu được khuyến khích dùng
  - Đặc biệt cho dữ liệu "lớn" như kiểu định nghĩa bởi lớp

Từ khóa const cho tham số

- Kiểu dữ liệu lớn (thường là kiểu định nghĩa bởi lớp)
  - Được khuyên là nên dùng truyền tham chiếu
  - Ngay cả khi hàm không biến đổi đối số
- Bảo vệ đối số
  - Dùng tham số hằng
    - Còn được gọi là tham số tham chiếu hằng
  - Đặt từ khóa const trước kiểu dữ liệu
  - Giới hạn tham số ở mức "chỉ đọc"
  - Nếu cố biến đổi tham số sẽ sinh lỗi biên dịch

Việc dùng từ khóa const

- Hoặc là tất cả hoặc không gì cả
- Nếu hàm không cần biến đổi dữ liệu
  - Bảo vệ tham số bằng const
  - Bảo vệ TẤT CẢ các tham số như vậy
- Điều này cũng áp dụng với tham số của hàm thành viên

Hàm inline

- Hàm không phải thành viên:
  - Dùng từ khóa inline trong khai báo hàm và dòng đầu định nghĩa hàm
- Hàm thành viên:
  - Đặt mã định nghĩa hàm BÊN TRONG định nghĩa lớp → tự động inline
- Chỉ dùng cho những hàm rất ngắn
- Mã thực sự sẽ được chèn vào nơi gọi
  - Bớt "chi phí" gọi hàm
  - Hiệu quả hơn, nhưng chỉ với hàm ngắn!

Hàm thành viên inline

- Định nghĩa hàm thành viên
  - Thường thì các định nghĩa được tách ra 1 tệp riêng
  - Có thể định nghĩa BÊN TRONG định nghĩa lớp
    - Khiến hàm này thành "in-line"
- Nhắc lại: chỉ dùng với những hàm rất ngắn
- Tính hiệu quả
  - Nếu quá dài → thật ra lại kém hiệu quả!

Thành viên static

- Biến thành viên static
  - Tất cả các đối tượng của lớp “dùng chung” một bản sao
  - Một đối tượng biến đổi nó → tất cả sẽ chịu tác động
- Hữu ích cho việc “theo dõi”
  - Một hàm thành viên được gọi bao nhiêu lần
  - Có bao nhiêu đối tượng đang tồn tại
- Trong khai báo, đặt từ khóa static trước kiểu

Hàm static

- Hàm thành viên có thể là static
  - Nếu không cần truy cập đến dữ liệu của đối tượng
  - Và vẫn "phải" là thành viên của lớp
  - Thì chỉ định nó là một hàm static
- Có thể được gọi từ bên ngoài lớp
  - Không qua đối tượng của lớp:
    - Ví dụ: Server::getTurn();
  - Thông qua đối tượng của lớp
    - Cách chuẩn: myObject.getTurn();
- Chỉ có thể sử dụng dữ liệu/hàm static!

Ví dụ thành viên static:
Display 7.6 Thành viên static (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;
```

Ví dụ thành viên static:
Display 7.6 Thành viên static (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
```

Ví dụ thành viên static:
Display 7.6 Thành viên static (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.*

Ví dụ thành viên static:
Display 7.6 Thành viên static (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

- Giới thiệu vector
  - Nhắc lại: mảng tĩnh có kích thước không đổi
  - vector: “là kiểu mảng có kích thước thay đổi”
    - trong quá trình thực thi chương trình
  - Định nghĩa trong Thư Viện Khuôn Mẫu Chuẩn - Standard Template Library (STL)
    - Dùng template lớp

Căn bản về vector

- Tương tự như mảng:
  - Có kiểu cơ sở cho từng phần tử
  - Lưu một tập các giá trị cùng thuộc kiểu cơ sở
- Nhưng cách khai báo thì khác:
  - Cú pháp: vector<Kiểu_Cơ_Sở>
    - Là dấu hiệu khuôn mẫu lớp
    - Bất cứ kiểu dữ liệu nào cũng có thể đặt vào vị trí Kiểu_Cơ_Sở
    - Tạo ra lớp vector "mới" cho kiểu cơ sở được chỉ định
  - Ví dụ khai báo:
    vector<int> v;

    so sanh voi int a[100];

Sử dụng vector

- vector<int> v;
  - "v là vector kiểu int"
  - Gọi tới hàm kiến tạo mặc định
    - tạo ra đối tượng vector rỗng
- Đánh chỉ số để truy cập giống như mảng
- Nhưng để thêm phần tử:
  - Phải gọi tới hàm thành viên push_back
- Hàm thành viên size()
  - Trả về số phần tử hiện thời

Ví dụ vector:
Display 7.7 Sử dụng 1 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      }
```

Ví dụ vector:
Display 7.7 Sử dụng 1 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

Tính hiệu quả của vector

- Hàm thành viên capacity()
  - Trả về dung lượng nhớ đã cấp phát cho đối tượng
  - Không giống size()
  - Thường thì capacity > size
    - Tự động tăng khi cần
- Nếu quan tâm đến tính hiệu quả:
  - Có thể thiết lập cách xử lý dung lượng thủ công
    - v.reserve(32); //đặt capacity bằng 32
    - v.reserve(v.size()+10); // đặt capacity bằng 10 cộng với size

Tóm tắt 1

- Hàm kiến tạo: tự động khởi tạo dữ liệu của lớp
  - Được gọi khi khai báo đối tượng
  - Hàm kiến tạo trùng tên với lớp
- Hàm kiến tạo mặc định không có đối số
  - Nên luôn định nghĩa
- Biến thành viên của lớp
  - Có thể là đối tượng của một lớp khác
    - Cần có phần khởi tạo trong hàm kiến tạo

Tóm tắt 2

- Tham số tham chiếu hằng
  - Hiệu quả hơn truyền giá trị
- Có thể viết inline những hàm rất ngắn
  - Có thể cải thiện hiệu quả
- Biến thành viên static
  - Được dùng chung bởi tất cả các đối tượng của lớp
- Các lớp vector
  - Giống như: “mảng có thể lớn lên hay nhỏ đi"

Chuẩn bị bài tới

- Đọc chương 8 giáo trình: Nạp chồng toán tử, Từ khóa friend và Tham chiếu