DỊCH SÁCH THE C++ PROGRAMMING LANGUAGE

CHƯƠNG 16

**Classes**

Một lớp là một kiểu do người dùng định nghĩa.

Các hàm thành viên có thể xác định ý nghĩa của khởi tạo (tạo), sao chép, di chuyển và dọn dẹp (phá hủy).

Các thành viên được truy cập bằng cách sử dụng **.**(dấu chấm) cho các đối tượng và **->**(mũi tên) cho con trỏ.

Các nhà khai thác, ví dụ như **+**, **!**, và **[]**, có thể được định nghĩa cho một lớp.

Một lớp là một không gian tên chứa các thành viên của nó.

Một **struct**là một **lớp**nơi các thành viên theo public.

Ví dụ:

**class X {**

**private:**

**int m;**

**public:**

**X(int i =0) :m{i} { }**

**int mf(int i)**

**{**

**int old = m;**

**m = i;**

**return old;**

**}**

**};**

**X var {7};**

**int user(X var, X ptr)**

**{**

**int x = var.mf(7);**

**int y = ptr−>mf(9);**

**int z = var.m;**

**}**

**16.2.1 Hàm thành viên**

Các *hàm thành viên* sử dụng cú pháp chuẩn để truy cập thành viên cấu trúc.

Ví dụ:

**Date my\_bir thday;**

**void f()**

**{**

**Date today;**

**today.init(16,10,1996);**

**my\_bir thday.init(30,12,1950);**

**Date tomorrow = today;**

**tomorrow.add\_day(1);**

**}**

Bởi vì các cấu trúc khác nhau có thể có các hàm thành viên có cùng tên, chúng ta phải chỉ định tên cấu trúc khi xác định một hàm thành viên.

**16.2.2 Sao chép mặc định**

Theo mặc định, các đối tượng có thể được sao chép. Đặc biệt, một đối tượng lớp có thể được khởi tạo bằng một bản sao của một đối tượng thuộc lớp của nó. Ví dụ:

**Date d1 = my\_bir thday;**

**Date d2 {my\_bir thday};**

Tương tự, theo mặc định, các đối tượng của lớp có thể được sao chép theo phép gán. Ví dụ:

**void f(Date& d)**

**{**

**d = my\_bir thday;**

**}**

**16.2.3 Kiểm soát truy cập**

Private: chỉ có thể được sử dụng bởi các hàm thành viên.

Public : *công khai*, tạo thành giao diện công khai cho các đối tượng của lớp.

Tuy nhiên, các chức năng nonmember bị cấm sử dụng các thành viên riêng tư. Ví dụ:

**void timewarp(Date& d)**

**{**

**d.y −= 200;** **//** *error : Date::y is private*

**}**

Có một số lợi ích thu được từ việc hạn chế quyền truy cập vào cấu trúc dữ liệu đối với danh sách các hàm được khai báo rõ ràng.

Bảo vệ dữ liệu riêng tư dựa trên việc hạn chế sử dụng tên thành viên trong lớp.

**16.2.4 Lớp và cấu trúc**

class **X {...};**

được gọi là *định nghĩa lớp*; nó định nghĩa một loại gọi là **X**.Một định nghĩa lớp có thể được sao chép trong các tệp nguồn khác nhau bằng cách sử dụng **#include**mà không vi phạm quy tắc một định nghĩa .

Theo định nghĩa, một **cấu trúc**là một lớp trong đó các thành viên được mặc định là công khai; đó là,

**struct S {/ \****...***\* /};**

chỉ đơn giản là viết tắt của

**class S {public: / \****...***\* /};**

Theo mặc định, các thành viên của một classlà riêng tư:

Không bắt buộc phải khai báo dữ liệu trước trong một class. Trên thực tế, thường hợp lý khi đặt các thành viên dữ liệu cuối cùng để nhấn mạnh các chức năng cung cấp giao diện người dùng công khai. Ví dụ:

**class Date3 {**

**public:**

**Date3(int dd, int mm, int yy);**

**void add\_year(int n);** **//** *add n years*

**private:**

**int d, m, y;**

**};**

Các chỉ định truy cập có thể được sử dụng nhiều lần trong một khai báo lớp duy nhất. Ví dụ:

**class Date4 {**

**public:**

**Date4(int dd, int mm, int yy);**

**private:**

**int d, m, y;**

**public:**

**void add\_year(int n);** **//** *add n years*

**};**

Tuy nhiên, có nhiều hơn một phần công khai, có xu hướng lộn xộn và có thể ảnh hưởng đến bố cục đối tượng .

**16.2.5 Hàm tạo**

Việc sử dụng các hàmđể cung cấp khởi tạo cho các đối tượng lớp là không phù hợp và dễ xảy ra lỗi. Bởi vì không có chỗ nào nói rằng một đối tượng phải được khởi tạo, một lập trình viên có thể quên làm như vậy - hoặc làm như vậy hai lần (thường cho kết quả thảm hại như nhau). Một cách tiếp cận tốt hơn là cho phép pro-grammer khai báo một hàm với mục đích rõ ràng là khởi tạo các đối tượng. Bởi vì một hàm như vậy xây dựng các giá trị của một kiểu nhất định, nó được gọi là một hàm *tạo*.

Hàm tạo có cùng tên với class.

Khi một lớp có một phương thức khởi tạo, tất cả các đối tượng của lớp đó sẽ được khởi tạo bởi một lời gọi phương thức khởi tạo. Nếu hàm tạo yêu cầu các đối số, các đối số này phải được cung cấp:

Vì một phương thức khởi tạo xác định quá trình khởi tạo cho một lớp, chúng ta có thể sử dụng ký hiệu **{}**:

Sử dụng ký hiệu **{} thay** ký hiệu **()**để khởi tạo vì nó rõ ràng về những gì đang được thực hiện (khởi tạo), tránh một số lỗi tiềm ẩn và có thể được sử dụng nhất quán. Có những trường hợp phải sử dụng ký hiệu **()**, nhưng chúng rất hiếm.

Bằng cách cung cấp một số hàm tạo, chúng tôi có thể cung cấp nhiều cách khác nhau để khởi tạo các đối tượng của một kiểu.

-Hàm tạo mặc định

-Hàm tạo không đầy đủ tham số

-Hàm tạo không đầy đủ tham số

-Hàm tạo sao chép

Ví dụ:

**class Date {**

**int d, m, y;**

**public:**

**//** *...*

**Date(int, int, int);**

**Date(int, int);**

**Date(int);**

**Date();**

**Date(const char );**

Với các hàm tạo, các hàm thành viên khác không còn phải đối phó với khả năng dữ liệu chưa được khởi tạo.

**16.2.6 Hàm tạo rõ ràng**

Một construc-tor được khai báo với từ khóa **explicit** chỉ có thể được sử dụng để khởi tạo và chuyển đổi rõ ràng. Ví dụ:

**struct X {**

**explicit X();**

**explicit X(int,int);**

|  |  |  |  |
| --- | --- | --- | --- |
| **};** |  |  |  |
| **X x1 = {};** | | | **//** *error: implicit* |
| **X x2** | **= {1,2};** | | **//** *error: implicit* |
| **X x3** | **{};** | | **//** *OK: explicit* |
| **X x4** | **{1,2};** | | **//** *OK: explicit* |
| **int f(X);** | | |  |
| **int i1 = f({});** | | | **//** *error: implicit* |
| **int i2** | | **= f({1,2});** | **//** *error: implicit* |
| **int i3** | | **= f(X{});** | **//** *OK: explicit* |
| **int i4** | | **= f(X{1,2});** | **//** *OK: explicit* |

Một khởi tạo với dấu **=**được coi là một *khởi tạo sao chép*.

Nếu bỏ dấu = thì được coi là khởi tạo trực tiếp.

Sự phân biệt giữa khởi tạo trực tiếp và sao chép được duy trì cho việc khởi tạo danh sách

**16.2.7 Bộ khởi tạo trong lớp**

Khi chúng ta sử dụng một số hàm tạo, việc khởi tạo thành viên có thể trở nên lặp đi lặp lại. Ví dụ:

**class Date {**

**int d, m, y;**

**public:**

**Date(int, int, int);**

**Date(int, int);**

**Date(int);**

**Date();**

**Date(const char );**

**//** *...*

**//** *day, month, year*

**//** *day, month, today’s year*

**//** *day, today’s month and year*

**//** *default Date: today*

**//** *date in string representation*

**};**

Chúng ta có thể giải quyết vấn đề đó bằng cách giới thiệu các đối số mặc định để giảm số lượng hàm tạo (§16.2.5). Ngoài ra, chúng ta có thể thêm trình khởi tạo vào các thành viên dữ liệu:

**class Date {**

**int d {today.d};**

**int m {today.m};**

**int y {today.y};**

**public:**

**Date(int, int, int);** **//** *day, month, year*

**Date(int, int);** **//** *day, month, today’s year*

**Date(int);** **//** *day, today’s month and year*

**Date();** **//** *default Date: today*

**Date(const char );** **//** *date in string representation*

**//** *...*

**16.2.8 Định nghĩa hàm trong lớp**

Một thành viên có thể tham chiếu đến một thành viên khác trong lớp của mình một cách độc lập với nơi thành viên đó được xác định (§6.3.4). ví dụ:

**class Date {**

**public:**

**void add\_month(int n) { m+=n; }** **//** *increment the Date’s m*

**private:**

**int d, m, y;**

**};**

**16.2.9 Thành viên tĩnh**

Các biến thành viên tĩnh là các biến thành viên mà thuộc về class, chứ không phải là thuộc về các đối tượng của class. Tương tự, một hàm cần truy cập vào các thành viên của một lớp, nhưng không cần phải được gọi cho một đối tượng cụ thể, được gọi là hàm thành viên **tĩnh**.

Nếu được sử dụng, một thành viên **tĩnh**- một hàm hoặc một thành viên dữ liệu - phải được định nghĩa ở đâu đó. Từ khóa **static**không được lặp lại trong định nghĩa của một thành viên **static**. Ví dụ:

**Date Date::default\_date {16,12,1770};**

**//** *definition of Date::default\_date*

**void Date::set\_default(int d, int m, int y)**

**{**

**//** *definition of Date::set\_default*

**default\_date = {d,m,y};**

**//** *assign new value to default\_date*

**}**

**16.2.10 Các loại thành viên**

Các kiểu và bí danh kiểu có thể là thành viên của một lớp. Ví dụ:

**template<typename T>**

**class Tree {**

**using value\_type = T;**

**//** *member alias*

**enum Policy { rb, splay, treeps }; class Node {**

**//** *member enum*

**//** *member class*

**Node right;**

**Node left;**

**value\_type value;**

**public:**

**void f(Tree );**

**};**

**Node top;**

**public:**

**void g(const T&);**

**//** *...*

**};**

Một lớp không có bất kỳ quyền truy cập đặc biệt nào đối với các thành viên của lớp lồng nhau của nó. Ví dụ:

**template<typename T>**

**void Tree::g(Tree::Node p)**

**{**

**value\_type val = right−>value;**

**//** *error : no object of type Tree::Node*

**value\_type v = p−>right−>value;**

**//** *error: Node::right is private*

**p−>f(this);**

**//** *OK*

**}**

**16.3 Lớp Trừu tượng**

**16.3.1 Chức năng thành viên**

Đương nhiên, một triển khai cho mỗi chức năng thành viên phải được cung cấp ở đâu đó. Ví dụ:

**Date::Date(int dd, Month mm, int yy)**

**:d{dd}, m{mm}, y{yy}**

**{**

**if (y == 0) y = default\_date().year();**

**if (m == Month{}) m = default\_date().month();**

**if (d == 0) d = default\_date().day();**

**if (!is\_valid()) throw Bad\_date();**

**}**

CHƯƠNG 17

**17,1** Hàm tạo và Hàm hủy

**17.2.1 Hàm tạo**

Một thành viên có cùng tên với lớp của nó được gọi là một phương thức *khởi tạo*. Ví dụ:

**class Vector {**

**public:**

**Vector(int s);**

**};**

Một khai báo phương thức khởi tạo chỉ định một danh sách đối số (chính xác như đối với một hàm) nhưng không có kiểu trả về. Tên của một lớp không thể được sử dụng cho một hàm thành viên thông thường, thành viên dữ liệu, kiểu thành viên, v.v., trong lớp. Ví dụ: 

**struct S {**

**S();**

**void S(int);**

**int S;**

**enum S { foo, bar };**

**//** *fine*

**//** *error : no type can be specified for a constructor*

**//** *error : the class name must denote a constructor*

**//** *error : the class name must denote a constructor*

**};**

**17.2.2 Hàm hủy và tài nguyên**

Một hàm tạo khởi tạo một đối tượng. Nói cách khác, nó tạo ra môi trường mà các chức năng thành viên hoạt động. Đôi khi, việc tạo ra môi trường đó liên quan đến việc lấy một tài nguyên - chẳng hạn như tệp, khóa hoặc một số bộ nhớ - phải được giải phóng sau khi sử dụng (§5.2, §13.3). Do đó, một số lớp cần một hàm được đảm bảo sẽ được gọi khi một đối tượng bị hủy theo cách tương tự như cách một phương thức khởi tạo được đảm bảo sẽ được gọi khi một đối tượng được tạo. Không thể tránh khỏi, một hàm như vậy được gọi là hàm *hủy*.

Tên của hàm hủy là **˜**theo sau là tên lớp, đối với exam-ple**˜Vector ()**. Một ý nghĩa của **˜**là '' phần bổ sung '' (§11.1.2), và hàm hủy cho một lớp hoàn thành các hàm tạo của nó. Một hàm hủy không nhận đối số và một lớp chỉ có thể có một bộ hủy cấu trúc.

Hàm hủy thường dọn dẹp và giải phóng tài nguyên. Ví dụ:

**class Vector {**

**public:**

**Vector(int s) :elem{new double[s]}, sz{s} { };** **//** *constructor: acquire memory*

**˜Vector() { delete[] elem; }** **//** *destructor: release memory*

1. *...*

**private:**

**double elem; //** *elem points to an array of sz doubles*

**int sz;** **//** *sz is non-negative*

**};**

Một cặp hàm tạo / hàm hủy phù hợp là cơ chế thông thường để thực hiện khái niệm về một đối tượng có kích thước thay đổi trong C ++.

Một lập trình viên khai báo hàm hủy cho một lớp cũng phải quyết định xem các đối tượng của lớp đó có thể được sao chép hoặc di chuyển hay không (§17.6).

**17.2.3 Tạo và hủy thành viên**

Bộ tạo và bộ hủy tương tác chính xác với cấu trúc phân cấp lớp (§3.2.4, Chương 20). Con-structor xây dựng một đối tượng lớp '' từ dưới lên '':

Đầu tiên, hàm tạo gọi các hàm tạo lớp cơ sở của nó,

Sau đó, nó gọi các hàm tạo thành viên, và cuối cùng, nó thực thi cơ thể của chính nó.

Đầu tiên, trình hủy thực thi phần thân của chính nó, sau đó, nó gọi các trình hủy thành viên của nó, và cuối cùng, nó gọi các hàm hủy lớp cơ sở của nó.

Nếu một lớp được sử dụng để cần một hàm tạo mặc định và nếu lớp đó không có các cấu trúc con khác, thì trình biên dịch sẽ cố gắng tạo một hàm tạo mặc định. Ví dụ:

**struct S1 {**

**string s;**

**};**

**S1 x;** **//** *OK: x.s is initialized to ""*

**17.2.4 Gọi hàm tạo và hủy**

Một hàm hủy được gọi ngầm khi thoát khỏi phạm vi hoặc bằng cách **xóa**. Thông thường không chỉ cần gọi rõ ràng một hàm hủy một cách rõ ràng; làm như vậy sẽ dẫn đến các lỗi khó chịu. Tuy nhiên, có những trường hợp hiếm hoi (nhưng quan trọng) mà một hàm hủy phải được gọi một cách rõ ràng.

**17.2.5 Bộ hủy ảo**

Một hàm hủy có thể được khai báo là **ảo**và thường phải dành cho một lớp có hàm ảo.

Ví dụ:

**class Shape {**

**public:**

**virtual void draw() = 0;**

**virtual ˜Shape();**

**};**

**class Circle {**

**public:**

**//** *...*

**void draw();**

**˜Circle();**

**//** *…*

**//** *overr ides ˜Shape()*

**};**

Lý do chúng ta cần một hàm hủy **ảo**là một đối tượng thường được thao tác thông qua giao diện được cung cấp bởi một lớp cơ sở cũng thường bị **xóa** thông qua giao diện đó:

**17.3** **Khởi tạo đối tượng lớp**

**17.3.1 Khởi tạo không có trình tạo**

Chúng ta không thể định nghĩa một phương thức khởi tạo cho một kiểu dựng sẵn, nhưng chúng ta có thể khởi tạo nó với một giá trị của kiểu phù hợp. Ví dụ:

Khởi tạo mặc định của việc sử dụng **{}**được định nghĩa là khởi tạo của từng thành viên bởi **{}**. Vì vậy, **không có nào**được khởi tạo thành **{{}, {}, {}}**, là **{"", "", 0}**(§17.3.3).

**17.3.2 Trình tạo mặc định**

Một phương thức khởi tạo có thể được gọi mà không cần đối số được gọi là một phương thức *khởi tạo mặc định*. Cấu trúc con mặc định rất phổ biến. Ví dụ:

 **class Vector {**

**public:**

**Vector(); //** *default constructor : no elements*

**};**

Một hàm tạo mặc định được sử dụng nếu không có đối số nào được chỉ định hoặc nếu danh sách bộ khởi tạo trống được cung cấp:

**Vectơ v1;** **//***OK*

**Vectơ v2 {};** **//***OK*

Một đối số mặc định (§12.2.5) có thể tạo một hàm tạo nhận các đối số vào một hàm tạo mặc định. Ví dụ:

**class String {**

**public:**

**String(const char p = ""); //** *default constructor : empty string* **//** *...*

**};**

**String s1;** **//** *OK*

**String s2 {};** **//** *OK*

**17,4 Sao chép và di chuyển**

Khi chúng ta cần chuyển một giá trị từ **a**sang **b**, chúng ta thường có hai lựa chọn khác nhau về mặt logic:

*Sao chép*là ý nghĩa quy ước của **x = y**; có nghĩa là, hiệu quả là các giá trị của **x**và **y**là cả hai bằng **y**‘s giá trị trước khi chuyển nhượng.

*Move*lá **x**với **y**‘s giá trị cũ và **y**với một số *di chuyển-từ nhà nước.*Đối với hầu hết các trường hợp lồng vào nhau, vùng chứa, trạng thái chuyển từ đó là '' trống ''.

Sự phân biệt logic đơn giản này bị nhầm lẫn bởi truyền thống và thực tế là chúng ta sử dụng cùng một ký hiệu cho cả di chuyển và sao chép.

Thông thường, một nước đi không thể ném, trong khi một bản sao có thể (vì nó có thể cần lấy tài nguyên), và một nước đi thường hiệu quả hơn một bản sao. Khi bạn viết một thao tác di chuyển, bạn nên để đối tượng nguồn ở trạng thái hợp lệ nhưng không xác định vì cuối cùng nó sẽ bị hủy và trình hủy không thể hủy đối tượng còn lại ở trạng thái không hợp lệ.

Để giúp chúng ta khỏi công việc lặp đi lặp lại tẻ nhạt, sao chép và di chuyển có các định nghĩa mặc định (§17.6.2).

**17.5.1 Sao chép**

Sao chép cho một lớp **X**được xác định bằng hai phép toán:

Sao chép hàm tạo: **X (const X &)**

Sao chép phép gán: **X & operator = (const X &)**

Bạn có thể xác định hai phép toán này bằng các loại đối số mạo hiểm hơn, chẳng hạn như **X dễ bay hơi &**, nhưng không; bạn sẽ chỉ nhầm lẫn chính bạn và những người khác. Một phương thức khởi tạo sao chép được cho là tạo một bản sao của một đối tượng mà không cần sửa đổi nó. Tương tự, bạn có thể sử dụng **const X &**làm kiểu trả về của gán bản sao.

Hàm tạo sao chép và phép gán sao chép khác nhau ở chỗ hàm tạo sao chép khởi tạo bộ nhớ chưa khởi tạo, trong khi toán tử gán bản sao phải xử lý chính xác một đối tượng đã được xây dựng và có thể sở hữu tài nguyên.

**17.5.1.3 Ý nghĩa của bản sao**

Một hàm tạo bản sao hoặc phép gán bản sao phải làm gì để được coi là '' một công cụ sao chép thích hợp ''? Ngoài việc được khai báo với một kiểu chính xác, một hoạt động sao chép phải có ngữ nghĩa sao chép thích hợp. Hãy xem xét một hoạt động sao chép, **x = y**, của hai đối tượng cùng loại. Để phù hợp với lập trình hướng giá trị nói chung (§16.3.4) và để sử dụng với thư viện chuẩn nói riêng (§31.2.2), hoạt động phải đáp ứng hai tiêu chí:

*Tính tương đương*: Sau **x = y**, các phép toán trên **x**và **y**sẽ cho kết quả giống nhau. Đặc biệt, nếu **==**được xác định cho kiểu của chúng, chúng ta sẽ có **x == y**và **f (x) == f (y)**cho bất kỳ hàm **f ()**nào chỉ phụ thuộc vào các giá trị của **x**và **y**(trái ngược với có hành vi của nó phụ thuộc vào địa chỉ của **x**và **y**).

*Độc lập*: Sau **x = y**, các phép toán trên **x**không được thay đổi hoàn toàn trạng thái của **y**, nghĩa là **f (x)**không thay đổi giá trị của **y**miễn là **f (x)**không tham chiếu đến **y**.

**17.5.2 Di chuyển**

Cách truyền thống để nhận giá trị từ **a**đến **b**là sao chép nó. Đối với một số nguyên trong mem-ory của máy tính, đó chỉ là điều duy nhất có ý nghĩa: đó là những gì phần cứng có thể làm với một lệnh duy nhất. Tuy nhiên, theo quan điểm chung và logic thì không phải như vậy. Hãy xem xét việc triển khai rõ ràng **swap ()**trao đổi giá trị của hai đối tượng. Ví dụ:

**template<class T> void swap(T& a, T& b) {**

**const T tmp = a;**

**a = b;**

**b = tmp;**

**};**

Vấn đề cơ bản là chúng ta thực sự không muốn sao chép gì cả: chúng ta chỉ muốn trao đổi các cặp giá trị.

Chúng ta cũng có thể nhìn vấn đề sao chép từ một quan điểm hoàn toàn khác: chúng ta thường không sao chép những thứ vật chất, trừ khi chúng ta hoàn toàn phải làm như vậy. Nếu bạn muốn mượn điện thoại của tôi, tôi sẽ chuyển điện thoại của tôi cho bạn chứ không phải biến đện thoại của tôi thành bản sao của riêng bạn. Nếu tôi cho bạn mượn xe của tôi, tôi sẽ đưa cho bạn một chìa khóa và bạn lái xe đi trong xe của tôi, chứ không phải trong bản sao mới tạo của bạn về chiếc xe của tôi. Một khi tôi đã đưa cho bạn một đối tượng, bạn có nó và tôi không còn làm nữa.