 [**Translated from English to Vietnamese - www.onlinedoctranslator.com**](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=pdf&utm_campaign=attribution)

**16**



***Những loại đó không phải là "trừu tượng";***

***chúng giống như thật*NS *và* flyến mạch*.***

***- Doug McIlroy***

* **Giới thiệu**
* **Kiến thức cơ bản về lớp học**

**Chức năng Thành viên; Sao chép mặc định; Kiểm soát truy cập;lớp và cấu trúc; Người xây dựng;rõ ràng Người xây dựng; Bộ khởi tạo trong lớp; Định nghĩa hàm trong lớp; Tính đột biến; Tự tham khảo; Quyền truy cập thành viên;tĩnh Các thành viên; Các loại thành viên**

**• Lớp bê tông**

**Chức năng Thành viên; Chức năng Người trợ giúp; Người vận hành quá tải; Tầm quan trọng của các lớp bê tông**



**• Lời khuyên**

**16.1 Giới thiệu**

**Các lớp C ++ là một công cụ để tạo các kiểu mới có thể được sử dụng thuận tiện như các kiểu tích hợp sẵn. Ngoài ra, các lớp dẫn xuất (§3.2.4, Chương 20) và các mẫu (§3.4, Chương 23) cho phép người lập trình thể hiện các mối quan hệ (hierachical và parametric) giữa các lớp và tận dụng các mối quan hệ đó.**

**Một kiểu là một đại diện cụ thể của một khái niệm (một ý tưởng, một khái niệm, v.v.). Ví dụ: kiểu tích hợp C ++ flyến mạch với các hoạt động của nó +, -, ∗, v.v., cung cấp một cách gần đúng cụ thể của khái niệm toán học về một số thực. Một lớp là một kiểu do người dùng định nghĩa. Chúng tôi thiết kế một kiểu mới để cung cấp định nghĩa về một khái niệm không có đối tác trực tiếp giữa các kiểu được tích hợp sẵn. Ví dụ: chúng tôi có thể cung cấp một loạiTrunk\_line trong một chương trình xử lý điện thoại, một loại Nổ cho một trò chơi điện tử hoặc một loại danh sách <Đoạn> cho một chương trình xử lý văn bản. Một chương trình cung cấp các loại phù hợp chặt chẽ với các khái niệm của ứng dụng có xu hướng dễ hiểu hơn, dễ lập luận hơn và dễ sửa đổi hơn một chương trình không có. Một tập hợp các loại do người dùng xác định được lựa chọn tốt**



**450** **Các lớp học** **Chương 16**

**cũng làm cho một chương trình ngắn gọn hơn. Ngoài ra, nó làm cho nhiều loại phân tích mã khả thi. Đặc biệt, nó cho phép trình biên dịch phát hiện việc sử dụng bất hợp pháp các đối tượng mà nếu không thì chỉ được tìm thấy thông qua kiểm tra toàn diện.**

* **tưởng cơ bản trong việc xác định một kiểu mới là tách các chi tiết ngẫu nhiên của việc triển khai (ví dụ: cách bố trí dữ liệu được sử dụng để lưu trữ một đối tượng của kiểu) khỏi các thuộc tính cần thiết cho việc sử dụng nó một cách chính xác (ví dụ: danh sách đầy đủ của các chức năng có thể truy cập dữ liệu). Sự tách biệt như vậy được thể hiện tốt nhất bằng cách phân luồng tất cả các mục đích sử dụng cấu trúc dữ liệu và các quy trình vệ sinh nội bộ của nó thông qua một giao diện cụ thể.**

**Chương này tập trung vào các kiểu tương đối đơn giản '' cụ thể '' do người dùng xác định mà về mặt logic không khác**

**nhiều so với các kiểu cài sẵn:**

***§16.2*** ***Kiến thức cơ bản về kinh điển* xâm nhập**

***§16.3*** ***Bê tông Classe***

**Các chương sau đi vào chi tiết hơn và trình bày các lớp trừu tượng và cấu trúc phân cấp lớp:**

***Chương 17*** ***Xây dựng, Dọn dẹp, Sao chép và Di chuyển* trình bày nhiều cách khác nhau để kiểm soát việc khởi**

**tạo các đối tượng của một lớp, cách sao chép và di chuyển các đối tượng cũng như cách cung cấp '' các hành động dọn dẹp '' sẽ được thực hiện khi một đối tượng bị phá hủy (ví dụ: ra khỏi phạm vi).**

***Chương 18*** ***Người vận hành quá tải* giải thích cách xác định toán tử đơn phân và nhị phân (chẳng hạnnhư +, ∗, và !) cho các loại do người dùng xác định và cách sử dụng chúng.*Nhà điều hành đặc***

***Chương 19*** ***biệt* xem xét cách xác định và sử dụng các toán tử (chẳng hạn như[],(),->,Mới) '' đặc biệt '' ởchỗ chúng thường được sử dụng theo những cách khác với các toán tử số học và logic. Đặc biệt, chương này chỉ ra cách định nghĩa một lớp chuỗi.**

***Chương 20*** ***Các lớp có nguồn gốc* giới thiệu các tính năng cơ bản của ngôn ngữ hỗ trợ lập trìnhhướng đối tượng. Các lớp cơ sở và dẫn xuất, các hàm ảo và kiểm soát truy cập được bao phủ.**

***Chương 21*** ***Cấu trúc phân cấp lớp* tập trung vào việc sử dụng các lớp cơ sở và lớp dẫn xuất để tổ chứchiệu quả**



**cống hiến t**

**kế thừa ple (các lớp có nhiều hơn một lớp cơ sở) cũng được bao phủ.*Thông tin loại thời* *Chương 22 gian chạy* mô tả các kỹ thuật để điều hướng rõ ràng các cấu trúc phân cấp lớp. Đặc**

**biệt, các thao tác chuyển đổi loại hìnhdynamic\_cast vàstatic\_cast được trình bày, cũng như hoạt động xác định kiểu của một đối tượng cho một trong các lớp cơ sở của nó (người đánh chữ).**

**16.2 Cơ bản về Class**

**Dưới đây là một bản tóm tắt rất ngắn gọn về các lớp:**

* **Một lớp là một kiểu do người dùng định nghĩa.**
* **Một lớp bao gồm một tập hợp các thành viên. Các loại thành viên phổ biến nhất là thành viên dữ liệu**

**và chức năng thành viên.**

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



**Mục 16.2** **Kiến thức cơ bản về kinh điển** **451**

* **Các thành viên được truy cập bằng . (dấu chấm) cho các đối tượng và -> (mũi tên) cho con trỏ.**
* **Các nhà khai thác, chẳng hạn như +, !, và [], có thể được định nghĩa cho một lớp.**
* **Một lớp là một không gian tên chứa các thành viên của nó.**
* **Các công cộng các thành viên cung cấp giao diện của lớp và riêng thành viên cung cấp chi tiết**

**thực hiện.**

**• MỘT cấu trúc là một lớp nơi các thành viên theo mặc định công cộng.**

**Ví dụ:**

**lớp X {**

**riêng:** **// *đại diện (triển khai) là riêng tư***

**int m;**

**công cộng:**

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

**int mf (int i)** **// *một chức năng thành viên***

**{**

**int cũ = m;**

**m = i;** **// *đặt một giá trị mới***

**trở lại cũ;** **// *trả lại giá trị cũ***

**}**

**};**

**X var {7}; //*một biến kiểu X, được khởi tạo thành 7***

**int user (X var, X∗ ptr) {**

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

**int y = ptr−> mf (9);**

**int z = var.m;**

**}**

* ***truy cập bằng cách sử dụng. (dấu chấm)*// *truy cập bằng -> (mũi tên)*// *lỗi: không thể truy cập thành viên riêng tư***



**Các phần sau đề cập đến các**

**ý tưởng, với các chi tiết p**

**16.2.1 Chức năng thành viên**

**Xem xét triển khai khái niệm ngày bằng cách sử dụng cấu trúc (§2.3.1, §8.2) để xác định biểu diễn của một Ngày và một tập hợp các hàm để thao tác với các biến kiểu này:**

**struct Ngày {** **// *đại diện***

**int d, m, y;**

**};**

**void init\_date (Ngày & d, int, int, int); void add\_year (Ngày & d, int n); void add\_month (Ngày & d, int n); void add\_day (Ngày & d, int n);**

* ***khởi tạo d***
* ***thêm n năm vào d*// *thêm n tháng vào d*// *thêm n ngày vào d***

**Không có kết nối rõ ràng nào giữa kiểu dữ liệu, Ngàyvà các chức năng này. Một kết nối như vậy có thể được thiết lập bằng cách khai báo các chức năng là thành viên:**



**452**

**Các lớp học**

**Chương 16**

**struct Ngày {**

**int d, m, y;**

**void init (int dd, int mm, int yy);**

**// *khởi tạo***

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

**void add\_month (int n);**

**// *thêm n năm***

**// *thêm n tháng***

**void add\_day (int n);**

**// *thêm n ngày***

**};**

**Các hàm được khai báo trong định nghĩa lớp (a cấu trúc là một loại giai cấp; §16.2.4) được gọi là*chức năng* *thành viên* và chỉ có thể được gọi cho một biến cụ thể của kiểu thích hợp bằng cách sử dụng cú phápchuẩn cho membe cấu trúc**

**Date my\_birthday;**

**void f ()**

**{**

**Ngày hôm nay;**

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

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

**Ngày mai = hôm nay;**

**ngày mai.add\_day (1);**

* ***...***

**}**

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

**void Date :: init (int dd, int mm, int yy) {**



**d = đ;**

**m = mm;**

**y = yy;**

**}**

**Trong một hàm thành viên, tên thành viên có thể được sử dụng mà không cần tham chiếu rõ ràng đến một đối tượng. Trong trường hợp đó, tên đề cập đến thành viên của đối tượng mà hàm được gọi. Ví dụ, khi**

**Ngày :: init () được gọi cho hôm nay, m = mm giao cho hôm nay.m.** **Mặt khác, khi thành**

**Ngày :: init () được gọi cho sinh nhật của tôi, m = mm giao cho my\_birthday.m.** **viên lớp A có chức năng**

**'' know '' nó đã được gọi cho đối tượng nào. Nhưng hãy xem §16.2.12 để biết khái niệm về mộttĩnh 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:**

**Ngày d1 = my\_birthday; Ngày d2 {my\_birthday};**

**// *khởi tạo bằng bản sao***

**// *khởi tạo bằng bản sao***



**Mục 16.2.2** **Sao chép mặc định** **453**

**Theo mặc định, bản sao của một đối tượng lớp là bản sao của mỗi thành viên. Nếu mặc định đó không phải là hành vi mong muốn cho một lớpNS, một hành vi thích hợp hơn có thể được cung cấp (§3.3, §17.5). Tương tự, theo mặc định, các**

**đối tượng lớp có thể được sao chép theo phép gán. Ví dụ:**

**void f (Ngày & d)**

**{**

**d = my\_birthday;**

**}**

**Một lần nữa, ngữ nghĩa mặc định là bản sao thành viên. Nếu đó không phải là lựa chọn phù hợp cho một lớp học NS, người dùng có thể xác định một toán tử gán thích hợp (§3.3, §17.5).**

**16.2.3 Kiểm soát truy cập**

**Tuyên bố của Ngày trong phần phụ trước cung cấp một tập hợp các hàm để thao tácNgày. Tuy nhiên, nó không chỉ rõ rằng những chức năng đó phải là những chức năng duy nhất phụ thuộc trực tiếp vàoNgàyđại diện của và những người duy nhất để truy cập trực tiếp các đối tượng của lớp Ngày. Hạn chế này có thể được thể hiện bằng cách sử dụnglớp Thay vì một cấu trúc:**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**void init (int dd, int mm, int yy);** **// *khởi tạo***

**void add\_year (int n);** **// *thêm n năm***

**void add\_month (int n);** **// *thêm n tháng***

**void add\_day (int n);** **// *thêm n ngày***

**};**

**Các công cộng nhãn phân tách nội dung lớp thành hai phần. Những cái tên trong phần đầu tiên,*riêng*, một phần chỉ có thể được sử dụng bởi các chức năng thành viên. Thư hai,*công cộng*, một phần cấu thành giao diện chung cho các đối tượng của lớp. MỘTcấu trúc là hàm si có thể được định nghĩa một**



**void Date :: add\_year (int n) {**

**y + = n;**

**}**

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

**void timewarp (Ngày & d)**

**{**

**dy - = 200;** **// *lỗi: Ngày :: y là riêng tư***

**}**

**Các trong đó() chức năng bây giờ là điều cần thiết vì việc đặt dữ liệu ở chế độ riêng tư buộc chúng tôi phải cung cấp cách khởi tạo các thành viên. Ví dụ:**

**Ngày dx;**

**dx.m = 3;** **// *error: m là riêng tư*//**

**dx.init (25,3,2011);** ***VÂNG***



**454** **Các lớp học** **Chương 16**

**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. Ví dụ: bất kỳ lỗi nào gây raNgày để nhận một giá trị bất hợp pháp (ví dụ: ngày 36 tháng 12 năm 2016) phải do mã trong một hàm thành viên gây ra. Điều này ngụ ý rằng giai đoạn đầu tiên của quá trình gỡ lỗi - bản địa hóa - được hoàn thành trước khi chương trình được chạy. Đây là một trường hợp đặc biệt của quan sát chung rằng bất kỳ thay đổi nào đối với hành vi của loạiNgày có thể và phải được thực hiện bởi những thay đổi đối với các thành viên của nó. Đặc biệt, nếu chúng ta thay đổi biểu diễn của một lớp, chúng ta chỉ cần thay đổi các hàm thành viên để tận dụng lợi thế của biểu diễn mới. Mã người dùng trực tiếp chỉ phụ thuộc vào giao diện công khai và không cần phải viết lại (mặc dù có thể cần phải biên dịch lại). Một ưu điểm khác là người dùng tiềm năng chỉ cần kiểm tra các định nghĩa của các hàm thành viên để học cách sử dụng một lớp. Một lợi thế tinh tế hơn, nhưng quan trọng nhất, là tập trung vào thiết kế**

**thời gian dành cho d**

**Việc bảo vệ dữ liệu riêng tư dựa trên việc hạn chế sử dụng tên thành viên trong lớp. Do đó, nó có thể bị phá vỡ bằng thao tác địa chỉ (§7.4.1) và chuyển đổi kiểu rõ ràng (§11.5). Nhưng điều này, tất nhiên, là gian lận. C ++ bảo vệ khỏi sự cố tình cờ thay vì cố tình gian lận (gian lận). Chỉ phần cứng mới có thể cung cấp khả năng bảo vệ hoàn hảo chống lại việc sử dụng ác ý một ngôn ngữ có mục đích chung và thậm chí điều đó khó có thể thực hiện được trong các hệ thống thực tế.**

**16.2.4 lớp và cấu trúc**

**Xay dung**

**lớp X {...};**

**được gọi là *định nghĩa lớp học*; nó định nghĩa một loại được gọi làNS. Vì lý do lịch sử, định nghĩa lớp thường được gọi là*khai báo lớp học*. Ngoài ra, giống như các khai báo không phải là định nghĩa, định nghĩa lớp có thể được sao chép trong các tệp nguồn khác nhau bằng cách sử dụng#bao gồm mà không vi phạm quy tắc một định nghĩa (§15.2.3).**

**Theo định nghĩa, a 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**

**lớp S {public: / \* *...* \* /};**

**Hai định nghĩa này của NS có thể hoán đổi cho nhau, mặc dù thường là khôn ngoan khi gắn bó với một kiểu. Bạn sử dụng phong cách nào là tùy thuộc vào hoàn cảnh và sở thích. Tôi có xu hướng sử dụngcấu trúc đối với các lớp mà tôi nghĩ là '' chỉ là cấu trúc dữ liệu đơn giản. '' Nếu tôi nghĩ về một lớp là '' một kiểu thích hợp với một bất biến, '' tôi sử dụnglớp. Các hàm tạo và các hàm truy cập có thể khá hữu ích ngay cả đối vớicấu trúcs, nhưng như một cách viết tắt chứ không phải là người bảo đảm sự bất biến (§2.4.3.2, §13.4).**

**Theo mặc định, các thành viên của một lớp là riêng tư:**

**lớp Ngày1 {**

**int d, m, y;**

* ***riêng tư theo mặc định***

**công cộng:**

**Date1 (int dd, int mm, int yy); void**

**add\_year (int n); //*thêm n năm***

**};**



**Mục 16.2.4** **lớp và cấu trúc** **455**

**Tuy nhiên, chúng tôi cũng có thể sử dụng công cụ xác định quyền truy cập riêng: để nói rằng các thành viên theo dõi là riêng tư,**

**giống như công cộng: nói rằng các thành viên sau đây là công khai:**

**struct Date2 {**

**riêng:**

**int d, m, y;**

**công cộng:**

**Date2 (int dd, int mm, int yy); void**

**add\_year (int n); //*thêm n năm***

**};**

**Ngoại trừ tên khác, Date1 và Date2 là tương đương.**

**Nó không phải là một yêu cầu cuối**

**cùng mà các thành viên nhấn mạnh**

**lớp Ngày3 {**

**công cộng:**

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

**add\_year (int n); //*thêm n năm***

**riêng:**

**int d, m, y;**

**};**

**Trong mã thực, nơi cả giao diện công khai và chi tiết triển khai thường rộng hơn so với trong các ví dụ hướng dẫn, tôi thường thích kiểu được sử dụng cho Date3.**

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

**lớp Ngày4 {**

**công cộng:**

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

**riêng:**

**int d, m, y;**



**công cộng:**

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

**};**

**Có nhiều hơn một phần công khai, như trong Ngày 4, tuy nhiên, có xu hướng lộn xộn và có thể ảnh hưởng đến bố cục đối tượng (§20.5). Vì vậy, không có nhiều hơn một phần riêng tư. Tuy nhiên, việc cho phép nhiều mã xác định truy cập trong một lớp rất hữu ích cho mã do máy tạo ra.**

**16.2.5 Cấu tạo**

**Việc sử dụng các chức năng như trong đó() để cung cấp khởi tạo cho các đối tượng lớp là không phù hợp và 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 lập trình viên 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*constructor*. Một phương thức khởi tạo được công nhận bởi có cùng tên với chính lớp đó. Ví dụ:**



**456** **Các lớp học** **Chương 16**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**Ngày (int dd, int mm, int** **// *constructor***

**yy); //*...***

**};**

**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:**

**Ngày hôm nay = Ngày (23,6,1983);**

**Ngày xmas (25,12,1990);**

**Date my\_birthday;**

**Ngày phát hành1\_0 (10,12);**

**Vì một phương thức khởi tạo định nghĩa việc khởi tạo cho một lớp, chúng ta có thể sử dụng {}ký hiệu -initializer:**

**Ngày hôm nay = Ngày {23,6,1983};**

**Ngày xmas {25,12,1990}; Ngày phát // *dạng viết tắt***

**hành1\_0 {10,12};** **// *lỗi: thiếu đối số thứ ba***

**Tôi khuyên bạn nên {} ký hiệu trên () 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 (§2.2.2, §6.3.5). Có những trường hợp() ký hiệu phải được sử dụng (§4.4.1, §17.3.2.1), 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. Ví dụ:**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**// *...***



**Ngày (int, int, int);**

**Ngày (int, int);**

**Ngày (int);**

**Ngày();** **// *Ngày mặc định: hôm nay*// *ngày***

**Ngày (const char∗);** ***trong biểu diễn chuỗi***

**};**

**Các hàm tạo tuân theo các quy tắc nạp chồng giống như các hàm thông thường (§12.3). Miễn là các hàm tạo có đủ khác biệt về kiểu đối số của chúng, trình biên dịch có thể chọn đúng để sử dụng:**

**Ngày hôm nay {4};** **// *4, hôm nay.m, hôm nay.y***

**Ngày tháng 7 {"4 tháng 7 năm 1983"}; Hẹn hò**

**với chàng trai {5,11};** **// *5, 11, today.y*// *mặc định khởi***

**Ngày bây giờ;** ***tạo như ngày hôm nay*// *mặc định***

**Ngày bắt đầu {};** ***khởi tạo như ngày hôm nay***

**Sự gia tăng của các nhà xây dựng trong Ngày ví dụ là điển hình.** **Khi thiết kế một lớp học, chuyên gia**

**grammer luôn bị cám dỗ để thêm các tính năng chỉ vì ai đó có thể muốn chúng. Cần suy nghĩ nhiều hơn để quyết định cẩn thận những tính năng nào thực sự cần thiết và chỉ bao gồm những tính năng đó. Tuy nhiên, suy nghĩ bổ sung đó thường dẫn đến các chương trình nhỏ hơn và dễ hiểu hơn. Một cách của**



**Mục 16.2.5** **Người xây dựng** **457**

**giảm số lượng các hàm liên quan là sử dụng các đối số mặc định (§12.2.5). VìNgày, mỗi đối số có thể được cung cấp một giá trị mặc định được hiểu là '' chọn giá trị mặc định: hôm nay. ''**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**Ngày (int dd = 0, int mm = 0, int yy =**

**0); //*...***

**};**

**Ngày :: Ngày (int dd, int mm, int yy) {**

**d = đ? dd: hôm nay.d**

**m = mm? mm: hôm nay.m; y**

**= yy? yy: hôm nay.y;**

* ***kiểm tra xem Ngày có hợp lệ không***

**}**

**Khi một giá trị đối số được sử dụng để biểu thị '' chọn giá trị mặc định '', giá trị được chọn phải nằm ngoài tập hợp các giá trị có thể có cho đối số. Vìngày và tháng, điều này rõ ràng là như vậy, nhưng đối với năm, số 0 có thể không phải là một lựa chọn hiển nhiên. May mắn thay, không có năm 0 trên lịch châu Âu; 1AD ( năm == 1) đến ngay sau 1BC (năm == - 1).**

**Ngoài ra, chúng tôi có thể sử dụng các giá trị mặc định trực tiếp làm đối số mặc định:**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**Ngày (int dd = today.d, int mm = today.m, int yy =**

**today.y); //*...***

**};**



**Ngày :: Ngày (int dd, int mm {**

* ***kiểm tra xem Ngày có hợp lệ không***

**}**

**Tuy nhiên, tôi đã chọn sử dụng 0 để tránh xây dựng các giá trị thực tế thành Ngàycủa giao diện. Bằng cách đó, chúng tôi có tùy chọn để cải thiện việc triển khai mặc định sau này.**

**Lưu ý rằng bằng cách đảm bảo việc khởi tạo các đối tượng một cách thích hợp, các hàm tạo sẽ đơn giản hóa rất nhiều việc thực hiện các hàm thành viên. Với các hàm tạo, các hàm thành viên khác không còn phải đối phó với khả năng dữ liệu chưa được khởi tạo (§16.3.1).**

**16.2.6 rõ ràng Người xây dựng**

**Theo mặc định, một hàm tạo được gọi bởi một đối số hoạt động như một chuyển đổi ngầm định từ kiểu đối số sang kiểu của nó. Ví dụ:**

**phức tạp <double> d {1};** **// *d == {1,0} (§5.6.2)***



**458** **Các lớp học** **Chương 16**

**Những chuyển đổi ngầm như vậy có thể cực kỳ hữu ích. Số phức là một ví dụ: nếu chúng ta bỏ đi phần ảo, chúng ta sẽ nhận được một số phức trên trục thực. Đó chính xác là những gì mà toán học yêu cầu. Tuy nhiên, trong nhiều trường hợp, những chuyển đổi như vậy có thể là một nguồn gây nhầm lẫn và sai sót đáng kể. Xem xétNgày:**

**void my\_fct (Ngày d);**

**void f ()**

**{**

**Ngày d {15};** **// *hợp lý: x trở thành {15, today.m, today.y}***

* ***...***

**my\_fct (15);**

**d = 15;**

* ***...***

**}**

* ***che khuất***
* ***che khuất***

**Tốt nhất, điều này là tối nghĩa. Không có kết nối logic rõ ràng giữa số 15 và một Ngày độc lập với sự phức tạp của mã của chúng tôi.**

**May mắn thay, chúng ta có thể chỉ định rằng một hàm tạo không được sử dụng làm *ngầm hiểu* sự chuyển đổi. Một hàm tạo được khai báo với từ khóarõ ràng chỉ có thể được sử dụng để khởi tạo và chuyển đổi rõ ràng. Ví dụ:**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**Ngày rõ ràng (int dd = 0, int mm = 0, int yy =**

**0); //*...***

**};**

**Ngày d1 {15};** **// *OK: được coi là rõ ràng*//**

**Ngày d2 = Ngày {15};** ***OK: rõ ràng***



**Ngày d3 = {15};**

**Ngày d4 = 15;**

**void f ()**

**{**

**my\_fct (15);** **// *lỗi: truyền đối số không thực hiện chuyển đổi ngầm định*//**

**my\_fct ({15});** ***lỗi: truyền đối số không thực hiện chuyển đổi ngầm định*// *OK:***

**my\_fct (Ngày {15});** ***rõ ràng***

**// *...***

**}**

**Khởi tạo với một = được coi là một *sao chép khởi tạo*. Về nguyên tắc, một bản sao của trình khởi tạo được đặt vào đối tượng được khởi tạo. Tuy nhiên, một bản sao như vậy có thể được tối ưu hóa (bỏ qua) và thao tác di chuyển (§3.3.2, §17.5.2) có thể được sử dụng nếu trình khởi tạo là một rvalue (§6.4.1). Bỏ ra ngoài= làm cho việc khởi tạo rõ ràng. Khởi tạo rõ ràng được gọi là*khởi tạo trực tiếp*. Theo mặc định, khai báo một**

**hàm tạo có thể được gọi với một đối sốrõ ràng. Bạn cần một**

**lý do chính đáng để không làm như vậy (đối với phức tạp). Nếu bạn xác định một hàm tạo ngầm, tốt nhất là ghi lại lý do của bạn hoặc người bảo trì có thể nghi ngờ rằng bạn đã quên (hoặc không biết gì).**



|  |  |  |
| --- | --- | --- |
| **Mục 16.2.6** | **rõ ràng Người xây dựng** | **459** |
| **Nếu một hàm tạo được khai báo rõ ràng và được định nghĩa bên ngoài lớp, rằng rõ ràng không thể lặp lại:** | |  |

**Ngày học {**

**int d, m, y;**

**công cộng:**

**Ngày rõ ràng (int dd);**

* ***...***

**};**

**Ngày :: Ngày (int dd) {/ \* *...* \* /} Ngày rõ** **// *VÂNG***

**ràng :: Ngày (int dd) {/ \* *...* \* /}** **// *lỗi***

**Hầu hết các ví dụ trong đó giải thíchcũng**

**có thể hữu ích cho việc xây dựng**

**struct X {**

**rõ ràng X ();**

**rõ ràng X (int, int);**

**};**

|  |  |  |  |
| --- | --- | --- | --- |
| **X x1 = {};** | | | **// *error: ẩn ý*** |
| **X x2** | **= {1,2};** | | **// *error: ẩn ý*** |
| **X x3** | **{};** | | **// *OK: rõ ràng*** |
| **X x4** | **{1,2};** | | **// *OK: rõ ràng*** |
| **int f (X);** | | |  |
| **int i1** | | **= f ({});** | **// *error: ẩn ý*** |
| **int i2** | | **= f ({1,2});** | **// *error: ẩn ý*** |
| **int i3** | | **= f (X {}); int** | **// *VÂNG*** |
| **i4 = f (X {1,2});** | | | **// *VÂNG*** |



**Sự khác biệt giữa khởi tạo trực tiếp và sao chép được duy trì cho việc khởi tạo danh sách (§17.3.4.3).**

**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 lại. Ví dụ:**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**Ngày (int, int, int);** **// *ngày tháng năm*// *ngày,***

**Ngày (int, int);** ***tháng, năm nay*// *ngày, tháng và***

**Ngày (int);** ***năm hôm nay*// *Ngày mặc định:***

**Ngày();** ***hôm nay*// *ngày trong biểu diễn***

**Ngày (const char∗** ***chuỗi***

**); //*...***

**};**



**460** **Các lớp học** **Chương 16**

**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 tôi có thể thêm trình khởi tạo vào các thành viên dữ liệu:**

**Ngày học {**

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

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

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

**công cộng:**

**Ngày (int, int, int);**

**Ngày (int, int);**

**Ngày (int);**

**// *ngày tháng năm*// *ngày,***

***tháng, năm nay*// *ngày, tháng***

***và năm hôm nay***

**Ngày();**

**Ngày (const char∗**

**); //*...***

**Bây giờ, mỗi hàm tạo có NS, NS, và y được khởi tạo trừ khi nó tự thực hiện. Ví dụ:**

**Ngày :: Ngày (int dd)**

**: d {đ}**

**{**

* ***kiểm tra xem Ngày có hợp lệ không***

**}**

**Điều này tương đương với:**

**Ngày :: Ngày (int dd)**

**: d {dd}, m {today.m}, y {today.y}**

**{**

* ***kiểm tra xem Ngày có hợp lệ không***

**}**

**16.2.8 Định nghĩa hàm trong lớp**



**Một hàm thành viên được xác**

**định là một nội tuyến (§12.1.5) m**

**các chức năng nhỏ, hiếm khi sửa đổi, được sử dụng thường xuyên. Giống như định nghĩa lớp mà nó là một phần, một hàm thành viên được định nghĩa trong lớp có thể được sao chép trong một số đơn vị dịch bằng cách sử dụng#bao gồm. Giống như bản thân lớp, ý nghĩa của hàm thành viên phải giống nhau cho dù nó ở đâu#bao gồmd (§15.2.3).**

**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). Xem xét:**

**Ngày học {**

**công cộng:**

**void add\_month (int n) {m + = n; } //*...***

**// *tăng ngày m***

**riêng:**

**int d, m, y;**

**};**

**Nghĩa là, các khai báo thành viên chức năng và dữ liệu là độc lập với thứ tự. Tôi có thể tương đương đã viết:**



**Ngày :: add\_month; chỉ thêmn và hy vọng đạt được một ngày tốt đẹp**

|  |  |  |
| --- | --- | --- |
| **Mục 16.2.8** | **Định nghĩa hàm trong lớp** | **461** |
| **Ngày học {** |  |  |
| **công cộng:** |  |  |
| **void add\_month (int n) {m + =** | **// *tăng ngày m*** |  |
| **n; } //*...*** |  |  |

**riêng:**

**int d, m, y;**

**};**

**nội tuyến void Date :: add\_month (int n) // *thêm n tháng*{**

**m + = n;** **// *người gia tăng***

**}**

**Kiểu sau này thường được sử dụng để giữ cho các định nghĩa lớp đơn giản và dễ đọc. Nó cũng cung cấp sự phân tách văn bản về giao diện và việc triển khai của một lớp.**

**Rõ ràng, tôi đã đơn giản hóa định nghĩa của là quá ngây thơ (§16.3.1).**

**16.2.9 Tính đột biến**

**Chúng ta có thể định nghĩa một đối tượng được đặt tên như một hằng số hoặc một biến. Nói cách khác, một cái tên có thể đề cập đến một đối tượng chứa*bất biến* hoặc một *có thể thay đổi* giá trị. Vì thuật ngữ chính xác có thể hơi vụng về, chúng tôi kết thúc việc đề cập đến một số biến là không đổi hoặc ngắn gọn vẫn làhăng sô biến. Tuy nhiên, điều này nghe có vẻ kỳ lạ đối với người nói tiếng Anh bản ngữ, nhưng khái niệm này lại hữu ích và được gắn sâu vào hệ thống loại C ++. Việc sử dụng có hệ thống các đối tượng không thay đổi dẫn đến mã dễ hiểu hơn, nhiều lỗi hơn được phát hiện sớm và đôi khi cải thiện hiệu suất. Đặc biệt, tính bất biến là đặc tính hữu ích nhất trong chương trình đa luồng (§5.3, Chương 41).**

**Để trở nên hữu ích ngoài định nghĩa về các hằng số đơn giản của các kiểu cài sẵn, chúng ta phải có khả năng xác định các hàm hoạt động trên hăng sô đối tượng của các kiểu do người dùng xác định. Đối với các chức năng tự do có nghĩa là các chức năng cóNScác chức năng thành viên hoạt động**

****

**16.2.9.1 Các chức năng thành viên không đổi**

**Các Ngày như được định nghĩa cho đến nay cung cấp các chức năng thành viên để đưa ra một Ngày một giá trị. Rất tiếc, chúng tôi không cung cấp cách kiểm tra giá trị củaNgày. Sự cố này có thể dễ dàng được khắc phục bằng cách thêm các chức năng đọc ngày, tháng và năm:**

**Ngày học {**

**int d, m, y;**

**công cộng:**

**int day () const {return d; } int**

**month () const {return m; } int**

**year () const;**

**void add\_year (int n);** **// *thêm n năm***

* ***...***

**};**

****

**MỘT**

**}**

**Nói cách khác,**

**trả lại y;**

**{**

**int Ngày :: năm ()**

**}**

**int Ngày :: năm () const**

**{**

**trả về ++ y;**

**Các hăng sô sau danh sách đối số (trống) trong khai báo hàm chỉ ra rằng các hàm này không sửa đổi trạng thái của Ngày.**

**Đương nhiên, trình biên dịch sẽ bắt gặp những nỗ lực vô tình vi phạm lời hứa này. Ví dụ:**

**// *error: cố gắng thay đổi giá trị thành viên trong hàm const***

**462** **Các lớp học** **Chương 16**

**Khi một hăng sô hàm thành viên được định nghĩa bên ngoài lớp của nó, hăng sô hậu tố là bắt buộc:**

**// *lỗi: thiếu const trong loại hàm thành viên***

**hăng sô là một phần của loại Ngày ngày(), Ngày :: tháng (), và Ngày :: năm ().**

**hăng sô hàm thành viên có thể được gọi cho cả hai hăng sô và khônghăng sô các đối tượng, trong khi một phi**

**hăng sô hàm thành viên chỉ có thể được gọi cho khônghăng sô các đối tượng. Ví dụ:**

**void f (Ngày & d, const Ngày & cd) {**

**int i = d.year ();** **// *VÂNG***

**d.add\_year (1);** **// *VÂNG***

**int j = cd.year ();** **// *VÂNG***

**cd.add\_year (1);** **// *lỗi: không thể thay đổi giá trị của một const Ngày***

**}**

**16.2.9.2 Hằng số vật lý và lôgic**

**Đôi khi, một hàm thành viên về mặt logic hăng sô, nhưng nó vẫn cần thay đổi giá trị của một thành viên. Đó là, đối với người dùng, f**

****

**người dùng không thể trực tiếp tuân theo Ngày lớp có thể có một hàm trả về một biểu diễn chuỗi. Việc xây dựng biểu diễn này có thể là một hoạt động tương đối tốn kém. Do đó, sẽ hợp lý nếu giữ một bản sao để các yêu cầu lặp đi lặp lại sẽ chỉ trả lại bản sao, trừ khiNgàygiá trị của đã được thay đổi. Việc lưu vào bộ nhớ đệm các giá trị như vậy phổ biến hơn đối với các cấu trúc dữ liệu phức tạp hơn, nhưng chúng ta hãy xem làm thế nào để đạt đượcNgày:**

**Ngày học {**

**công cộng:**

**// *...***

**chuỗi string\_rep () const;** **// *biểu diễn chuỗi***

**riêng:**

**bool cache\_valid;**

**bộ nhớ cache chuỗi;**

**void compute\_cache\_value (); //*fill bộ nhớ cache***

* ***...***

**};**

****

**Mục 16.2.9.2** **Hằng số vật lý và lôgic** **463**

**Theo quan điểm của người dùng, string\_rep không thay đổi trạng thái của nó Ngày, vì vậy rõ ràng nó phải là mộthăng sô chức năng thành viên. Mặt khác,bộ nhớ đệm và cache\_valid các thành viên thỉnh thoảng phải thay đổi để thiết kế có ý nghĩa.**

**Những vấn đề như vậy có thể được giải quyết thông qua bạo lực bằng cách sử dụng một diễn viên, ví dụ, const\_cast**

**(§11.5.2). Tuy nhiên, cũng có những giải pháp thanh lịch hợp lý mà không liên quan đến việc gây rối với các quy tắc loại.**

**16.2.9.3 có thể thay đổi**

**Chúng ta có thể xác định một thành viên của một lớp là có thể thay đổi, nghĩa là nó có thể được sửa đổi ngay cả trong hăng sôsự vật:**

**Ngày học {**

**công cộng:**

**// *...***

**chuỗi string\_rep () const;** **// *biểu diễn chuỗi***

**riêng:**

**bool cache\_valid có thể thay đổi; bộ**

**đệm chuỗi có thể thay đổi;**

**void compute\_cache\_value ()**

**const; //*...***

**};**

* ***fill (có thể thay đổi) bộ nhớ cache***

**Bây giờ chúng ta có thể xác định string\_rep () theo cách rõ ràng:**

**chuỗi Ngày :: string\_rep () const {**

**if (! cache\_valid) {**

**compute\_cache\_value ();**

**cache\_valid = true;**

**}**



**trả lại bộ nhớ cache;**

**}**

**Bây giờ chúng ta có thể sử dụng string\_rep () cho cả hai hăng sô và khônghăng sô các đối tượng. Ví dụ:**

**void f (Ngày d, const Ngày cd) {**

**string s1 = d.string\_rep ();**

**string s2 = cd.string\_rep (); //** **// *VÂNG!***

***...***

**}**

**16.2.9.4 Tính đột biến thông qua hướng dẫn**

**Khai báo thành viên có thể thay đổi là thích hợp nhất khi chỉ một phần nhỏ của hình biểu diễn của một đối tượng nhỏ được phép thay đổi. Các trường hợp phức tạp hơn thường được xử lý tốt hơn bằng cách đặt dữ liệu thay đổi vào một đối tượng riêng biệt và truy cập nó một cách gián tiếp. Nếu kỹ thuật đó được sử dụng, ví dụ stringwith-cache sẽ trở thành:**



**464** **Các lớp học** **Chương 16**

**cấu trúc bộ nhớ cache {**

**bool hợp lệ;**

**chuỗi đại diện;**

**};**

**Ngày học {**

**công cộng:**

**// *...***

**chuỗi string\_rep () const;** **// *biểu diễn chuỗi***

**riêng:**

**bộ nhớ đệm∗ NS;**

**void compute\_cache**

**// *...***

**};**

**chuỗi Ngày :: string\_rep () const {**

**if (! c−> hợp lệ) {**

**compute\_cache\_value ();**

**c−> valid = true;**

**}**

**return c−> rep;**

**}**

**Các kỹ thuật lập trình hỗ trợ bộ đệm ẩn chung chung cho các hình thức đánh giá lười biếng khác nhau.**

**Lưu ý rằng hăng sô không áp dụng (tạm thời) cho các đối tượng được truy cập thông qua con trỏ hoặc tham chiếu. Người đọc con người có thể coi một đối tượng đó là '' một loại đối tượng subobject '', nhưng trình biên dịch không biết các con trỏ hoặc tham chiếu đó có gì khác với bất kỳ đối tượng nào khác hay không. Có nghĩa là, một con trỏ thành viên không có bất kỳ ngữ nghĩa đặc biệt nào để phân biệt nó với các con trỏ khác.**



**16.2.10 Tự giới thiệu**

**Các chức năng cập nhật trạng thái**

**trả về giá trị. Đối với một tập hợp các chức năng cập nhật liên quan như vậy, thường hữu ích khi trả về một tham chiếu đến đối tượng được cập nhật để các hoạt động có thể được xâu chuỗi. Ví dụ, chúng tôi muốn viết:**

**void f (Ngày & d)**

**{**

* ***...***

**d.add\_day (1) .add\_month (1) .add\_year (1); //**

***...***

**}**

**để thêm một ngày, một tháng và một năm vào NS. Để làm điều này, mỗi hàm phải được khai báo để trả về một tham chiếu đến mộtNgày:**

**Ngày học {**

**// *...***



|  |  |  |
| --- | --- | --- |
| **Mục 16.2.10** | **Tự tham khảo** | **465** |
| **Ngày & add\_year (int n);** | **// *thêm n năm*** |  |
| **Ngày & add\_month (int n); //*thêm n tháng*** | |  |
| **Ngày & add\_day (int n); //*thêm n ngày*** | |  |

**};**

**Mỗi (không-tĩnh) hàm thành viên biết đối tượng nào nó được gọi và có thể tham chiếu rõ ràng đến nó. Ví dụ:**

**Ngày & tháng :: add\_year (int n) {**

**if (d == 29 && m == 2 &&! leapyear (y + n)) {// *cẩn thận ngày 29 tháng 2***

**d = 1;**

**m = 3;**

**}**

**y + = n;**

**trở lại ∗cái này;**

**}**

**Cách diễn đạt ∗cái này đề cập đến đối tượng mà một hàm thành viên được gọi.**

**Trong một khôngtĩnh chức năng thành viên, từ khóa cái này là một con trỏ tới đối tượng mà hàm được gọi. Trong một khônghăng sô chức năng thành viên của lớp NS, loại cái này Là NS∗. Tuy vậy,cái này được coi là rvalue, vì vậy không thể lấy địa chỉ của cái này hoặc để gán cho cái này. Trong mộthăng sô chức năng thành viên của lớp NS, loại cái này Là const X∗ để ngăn chặn việc sửa đổi chính đối tượng (xem thêm §7.5).**

**Hầu hết việc sử dụng cái này được ngầm hiểu. Đặc biệt, mọi tham chiếu đến khôngtĩnh thành viên từ bên trong một lớp dựa vào việc sử dụng ngầm định cái này để có được thành viên của đối tượng thích hợp. Ví dụ,thêm năm Hàm có thể tương đương, nhưng thật tệ, đã được định nghĩa như thế này:**

**Ngày & tháng :: add\_year (int n) {**

**if (this−> d == 29 && this−> m == 2 &&! leapyear (this−> y + n)) {**



**this−> d = 1;**

**this−> m = 3;**

**}**

**this−> y + = n;**

**trở lại ∗cái này;**

**}**

**Một cách sử dụng rõ ràng phổ biến của cái này đang trong thao tác danh sách liên kết. Ví dụ:**

**liên kết struct {**

**Liên kết∗ trước;**

**Liên kết∗ suc khoe;**

**dữ liệu int;**

**Liên kết∗ insert (int x) {** **// *chèn x trước cái này***

**return pre = new Liên kết {pre, this, x};**

**}**



**466** **Các lớp học** **Chương 16**

**void remove () // *loại bỏ và phá hủy cái này*{**

**if (pre) pre−> suc = suc; if**

**(suc) suc−> pre = pre; xóa**

**cái này đi;**

**}**

* ***...***

**};**

**Sử dụng rõ ràng cái này được yêu cầu để truy cập vào các thành viên của các lớp cơ sở từ một lớp dẫn xuất là một khuôn mẫu (§26.3.7).**

**16.2.11 Quyền truy cập thành viên**

**Một thành viên của một lớp học NS có thể được truy cập bằng cách áp dụng . (dấu chấm) toán tử cho một đối tượng của lớp NS hoặc bằng cách áp dụng -> (mũi tên) toán tử đến một con trỏ đến một đối tượng của lớp NS. Ví dụ:**

**struct X {**

**void f ();**

**int m;**

**};**

**người dùng vô hiệu (X x, X∗ px)**

**{**

**m = 1;** **// *lỗi: không có m trong phạm vi*//**

**xm = 1;** ***VÂNG***

**x−> m = 1;** **// *lỗi: x không phải là một con trỏ*//**

**px−> m = 1;** ***VÂNG***

**px.m = 1;** **// *error: px là một con trỏ***

**}**



**Rõ ràng, có một chút liên quan đến**

**NS∗, vì vậy một vở kịch duy nhất**

**hợp nhất, vì vậy từ những ngày đầu tiên của C, quy tắc đã được sử dụng các toán tử riêng biệt.**

**Từ bên trong một lớp không cần toán tử. Ví dụ:**

**void X :: f ()**

**{**

**m = 1;** **// *Được: '' this-> m = 1; '' (§16.2.10)***

**}**

**Đó là, một tên thành viên không đủ điều kiện hoạt động như thể nó đã có tiền tố là this−>. Lưu ý rằng một hàm thành viên có thể tham chiếu đến tên của một thành viên trước khi nó được khai báo:**

**struct X {**

**int f () {return m; } //*fine: trả lại X này m*int**

**m;**

**};**

**Nếu chúng ta muốn tham chiếu đến một thành viên nói chung, thay vì một thành viên của một đối tượng cụ thể, chúng ta đủ điều kiện theo tên lớp theo sau là ::. Ví dụ:**



|  |  |  |
| --- | --- | --- |
| **Mục 16.2.11** | **Quyền truy cập thành viên** | **467** |
| **struct S {** |  |  |
| **int m;** |  |  |
| **int f ();** |  |  |
| **tĩnh int sm;** |  |  |
| **};** |  |  |
| **int X :: f () {return m; }** | **// *X của f*** |  |
| **int X :: sm {7};** | **// *Thành viên tĩnh của X sm (§16.2.12)*//** |  |
| **int (S ::∗) pmf () {& S :: f};** | ***Thành viên của X*** |  |

**Cấu trúc cuối cùng đó (một con trỏ đến thành viên) khá hiếm và bí truyền; xem §20.6. Tôi đề cập đến nó ở đây chỉ để nhấn mạnh tính tổng quát**

**16.2.12 [tĩnh] Thành viên**

**Sự tiện lợi của giá trị mặc định cho Ngàys đã được mua với cái giá phải trả là một vấn đề tiềm ẩn đáng kể. Của chúng tôiNgày lớp trở nên phụ thuộc vào biến toàn cục hôm nay. Cái nàyNgày lớp chỉ có thể được sử dụng trong một ngữ cảnh trong đó hôm nay được định nghĩa và sử dụng chính xác bởi mọi đoạn mã. Đây là loại ràng buộc khiến một lớp trở nên vô dụng bên ngoài ngữ cảnh mà nó được viết lần đầu tiên. Người dùng nhận được quá nhiều bất ngờ khó chịu khi cố gắng sử dụng các lớp phụ thuộc ngữ cảnh như vậy và việc bảo trì trở nên lộn xộn. Có thể '' chỉ một biến toàn cục nhỏ '' không quá khó quản lý, nhưng kiểu đó dẫn đến mã vô dụng ngoại trừ lập trình viên ban đầu của nó. Nó nên được tránh.**

**May mắn thay, chúng ta có thể có được sự tiện lợi mà không có sự cản trở của một biến toàn cầu có thể truy cập công khai. Một biến là một phần của một lớp, nhưng không phải là một phần của một đối tượng của lớp đó, được gọi làtĩnh thành viên. Có chính xác một bản sao của mộttĩnh thành viên thay vì một bản sao cho mỗi đối tượng, như đối vớitĩnh thành viên (§6.4.2). Tương tự, một hàm cần quyền truy cập vào các thành viên của một lớp, nhưng không cần được gọi cho một đối tượng cụ thể, được gọi làtĩnh chức năng thành viên.**

**Đây là một thiết kế lại bảo tồn ngữ nghĩa của các giá trị hàm tạo mặc định cho Ngày mà không có các vấn đề bắt nguồn từ sự phụ thuộc vào toàn cầu:**



**Ngày học {**

**int d, m, y;**

**static Date default\_date;**

**công cộng:**

**Ngày (int dd = 0, int mm = 0, int yy =**

**0); //*...***

**static void set\_default (int dd, int mm, int yy); //*đặt default\_date thành Date (dd, mm, yy)***

**};**

**Bây giờ chúng ta có thể xác định Ngày nhà xây dựng để sử dụng default\_date như thế này:**

**Ngày :: Ngày (int dd, int mm, int yy) {**

**d = đ? dd: default\_date.d;**

**m = mm? mm: default\_date.m; y**

**= yy? yy: default\_date.y;**

* ***... kiểm tra xem Ngày có hợp lệ không ...***

**}**



**468** **Các lớp học** **Chương 16**

**Sử dụng set\_default (), chúng tôi có thể thay đổi ngày mặc định khi thích hợp. MỘTtĩnh thành viên có thể được giới thiệu như bất kỳ thành viên nào khác. Trong additionatĩnh thành viên có thể được giới thiệu mà không đề cập đến một đối tượng. Thay vào đó, tên của nó đủ điều kiện theo tên của lớp của nó. Ví dụ:**

**void f ()**

**{**

**Ngày :: set\_default (4,5,1945);** **// *gọi thành viên tĩnh của Date set\_default ()***

**}**

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

**Ngày Date :: default\_date {1**

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

**// *định nghĩa của Date :: set\_default***

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

**// *gán giá trị mới cho default\_date***

**}**

**Bây giờ, giá trị mặc định là ngày sinh của Beethoven - cho đến khi ai đó quyết định khác.**

**Lưu ý rằng Ngày{} phục vụ như một ký hiệu cho giá trị của Ngày :: default\_date. Ví dụ:**

**Date copy\_of\_default\_date = Ngày {};**

**void f (Ngày);**

**void g ()**

**{**

**f (Ngày {});**

**}**

**Do đó, chúng tôi không cần một chức năng riêng biệt để đọc ngày mặc định. Hơn nữa, nơi loại đích rõ ràng làNgày, trơn {} là đủ. Ví dụ:**



**void f1 (Ngày);**

**void f2 (Ngày);**

**void f2 (int);**

**void g ()**

**{**

**f1 ({});**

**// *OK: tương đương với f1 (Ngày {})*// *lỗi: không***

**f2 ({}):**

***rõ ràng: f2 (int) hoặc f2 (Ngày)?*// *VÂNG***

**f2 (Ngày {});**

**Trong mã đa luồng, tĩnh các thành viên dữ liệu yêu cầu một số loại kỷ luật khóa hoặc truy cập để tránh các điều kiện chủng tộc (§5.3.4, §41.2.4). Vì đa luồng hiện nay rất phổ biến nên việc sử dụngtĩnh thành viên dữ liệu đã khá phổ biến trong mã cũ hơn. Mã cũ hơn có xu hướng sử dụngtĩnhcác thành viên theo cách ngụ ý các điều kiện chủng tộc.**



**Mục 16.2.13** **Các loại thành viên** **469**

**16.2.13 Các loại thành viên**

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

**mẫu <typename T>**

**cây lớp {**

**using value\_type = T;**

**// *bí danh thành viên***

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

**// *thành viên enum***

**Nút lớp {**

**// *lớp thành viên***

**Nút∗ đúng;**

**Nút∗ bên trái;**

**value\_type value;**

**công cộng:**

**void f (Cây∗);**

**};**

**Nút∗ đứng đầu;**

**công cộng:**

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

* ***...***

**};**

**MỘT *lớp thành viên* (thường được gọi là *lớp lồng nhau*) có thể tham khảo các loại và tĩnh các thành viên của lớp bao quanh của nó. Nó chỉ có thể đề cập đến khôngtĩnh các thành viên khi nó được cung cấp một đối tượng của lớp bao quanh để tham chiếu đến. Để tránh đi vào sự phức tạp của cây nhị phân, tôi sử dụng kỹ thuật thuần túy ''NS() vàNS()ví dụ về kiểu ''.**

**Một lớp lồng nhau có quyền truy cập vào các thành viên của lớp bao quanh nó, thậm chí riêng các thành viên (giống như một hàm thành viên có), nhưng không có khái niệm về một đối tượng hiện tại của lớp bao quanh. Ví dụ:**

**mẫu <typename T>**

**void Tree :: Node :: f (Tree∗ P) {**



**trên cùng = bên phải;** **// *lỗi: không có đối tượng của loại Cây được chỉ định***

**p−> top = right;**

**value\_type v = left−>**

**}**

**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:**

**mẫu <typename T>**

**void Tree :: g (Tree :: Node∗ P) {**

**value\_type val = right−> value;**

**// *lỗi: không có đối tượng của loại Tree ::***

**value\_type v = p−> right−> value; p**

***Node*// *lỗi: Node :: right là riêng tư*// *VÂNG***

**−> f (this);**

**}**

**Các lớp thành viên là một sự tiện lợi về mặt ký hiệu hơn là một tính năng có tầm quan trọng cơ bản. Mặt khác, bí danh thành viên rất quan trọng như là cơ sở của kỹ thuật lập trình chung dựa trên các kiểu liên kết (§28.2.4, §33.1.3). Thành viênenums thường là một sự thay thế cho lớp enum tránh làm ô nhiễm phạm vi bao quanh với tên của các điều tra viên (§8.4.1).**



**470** **Các lớp học** **Chương 16**

**16.3 Lớp bê tông**

**Phần trước đã thảo luận về các bit và phần của thiết kế Ngày lớp trong bối cảnh giới thiệu các tính năng ngôn ngữ cơ bản để xác định các lớp. Ở đây, tôi đảo ngược sự nhấn mạnh và thảo luận về việc thiết kế mộtNgày và cho biết các tính năng ngôn ngữ hỗ trợ thiết kế này như thế nào.**

**Các trừu tượng nhỏ, được sử dụng nhiều thường phổ biến trong nhiều ứng dụng. Ví dụ như ký tự Latinh, ký tự Trung Quốc, số nguyên, số dấu phẩy động, số phức, điểm, con trỏ, tọa độ, phép biến đổi, (*con trỏ, bù đắp*) cặp, ngày, giờ, phạm vi, liên kết, liên kết, nút, (*giá trị, đơn vị***

* **cặp, vị trí đĩa, vị trí mã nguồn, giá trị tiền tệ, đường thẳng, hình chữ nhật, số điểm cố định được chia tỷ lệ, số**

**tion sử dụng một số trong số này.**

**ứng dụng sử dụng một số trực tiếp và nhiều ứng dụng gián tiếp từ các thư viện.**

**C ++ hỗ trợ trực tiếp một số trừu tượng này dưới dạng các kiểu tích hợp sẵn. Tuy nhiên, hầu hết đều không và không thể được ngôn ngữ hỗ trợ trực tiếp vì có quá nhiều ngôn ngữ đó. Hơn nữa, người thiết kế một ngôn ngữ lập trình có mục đích chung không thể thấy trước nhu cầu chi tiết của mọi ứng dụng. Do đó, các cơ chế phải được cung cấp cho người sử dụng để xác định các loại bê tông nhỏ. Những loại như vậy được gọi là*các loại bê tông* hoặc *lớp bê tông* để phân biệt chúng với các lớp trừu tượng (§20.4) và các lớp trong phân cấp lớp (§20.3, §21.2).**

**Một lớp được gọi là *bê tông* (hoặc *một lớp cụ thể*) nếu biểu diễn của nó là một phần trong định nghĩa của nó. Cái này**

**phân biệt nó với các lớp trừu tượng (§3.2.2, §20.4) cung cấp giao diện cho nhiều cách triển khai khác nhau. Có sẵn đại diện cho phép chúng tôi:**

* **Để đặt các đối tượng trên ngăn xếp, trong bộ nhớ được cấp phát tĩnh và trong các đối tượng khác**
* **Để sao chép và di chuyển các đối tượng (§3.3, §17.5)**
* **Để tham chiếu trực tiếp đến các đối tượng được đặt tên (trái ngược với việc truy cập thông qua con trỏ và tham chiếu)**

**Điều này làm cho các lớp cụ thể trở nên đơn giản để lập luận và dễ dàng để trình biên dịch tạo ra mã tối ưu. Do đó, chúng tôi ưu tiên các lớp cụ thể cho các loại nhỏ, được sử dụng thường xuyên và quan trọng về hiệu suất, chẳng hạn như số phức (§5.6.2), con trỏ thông minh (§5.2.1) và vùng chứa (§4.4).**

**Đó là một loại được xác định**



**rõ ràng ban đầu rất tốt. Thứ tự**

**trần tục có ý nghĩa thống kê hơn nhiều so với phức tạp và tinh vi. Dưới ánh sáng này, chúng ta hãy xây dựng mộtNgày lớp:**

**không gian tên Chrono {**

**enum class Tháng {jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec};**

**Ngày học {**

**công cộng:** **// *giao diện công khai:***

**lớp Bad\_date {}; //*lớp ngoại lệ***

**Ngày rõ ràng (int dd = {}, Tháng mm = {}, int yy = {});** **// *{} có nghĩa là '' chọn một mặc định ''***



**Mục 16.3** **Lớp bê tông** **471**

* ***các chức năng không sửa đổi để kiểm tra Ngày:* int day () const;**

**Tháng tháng () const; int năm () const;**

**chuỗi string\_rep () const;** **// *biểu diễn chuỗi*// *Biểu diễn***

**void char\_rep (char s [], in max) const;** ***chuỗi kiểu C***

* ***(sửa đổi) các chức năng để thay đổi Ngày:***

**Ngày & add\_year (int n);** **// *thêm n năm***

**Ngày & add\_month (int n);** **// *thêm n tháng***

**Ngày & thêm\_ngày**

**riêng:**

**bool is\_valid ();**

**int d, m, y;**

* ***kiểm tra xem Ngày này có đại diện cho một ngày không*// *đại diện***

**};**

|  |  |  |
| --- | --- | --- |
| **bool is\_date (int d, Tháng m, int y);** | **// *đúng với ngày hợp lệ*//** | |
| **bool is\_leapyear (int y);** | ***đúng nếu y là năm nhuận*** | |
| **toán tử bool == (const Ngày & a, const Ngày & b);** |  |  |
| **toán tử bool! = (const Ngày & a, const Ngày & b);** |  |  |
| **const Ngày & default\_date ();** | **// *ngày mặc định*** | |
| **ostream & operator << (ostream & os, const Date & d);** |  | **// *in d tới hệ điều hành*// *đọc*** |
| **istream & operator >> (istream & is, Date & d); } //*Chrono*** | | ***Ngày từ là thành d*** |

**Tập hợp các thao tác này khá điển hình cho kiểu do người dùng xác định:**

1. **Một phương thức khởi tạo chỉ định cách khởi tạo các đối tượng / biến của kiểu (§16.2.5).**
2. **Một tập hợp các hàm**



**để chỉ ra rằng t**

**được gọi.**

1. **Một tập hợp các chức năng cho phép người dùng sửa đổi Ngàys mà không thực sự phải biết các chi tiết của biểu diễn hoặc tìm hiểu về sự phức tạp của ngữ nghĩa.**
2. **Các hoạt động được xác định rõ ràng cho phép Ngàyđược sao chép tự do (§16.2.2).**
3. **Một lớp học, Ngày tồi tệ, được sử dụng để báo cáo lỗi như là trường hợp ngoại lệ.**
4. **Một tập hợp các chức năng trợ giúp hữu ích. Các chức năng trợ giúp không phải là thành viên và không có quyền**

**truy cập trực tiếp vào đại diện củaNgày, nhưng chúng được xác định là có liên quan bằng cách sử dụng không gian tên Chrono.**

**Tôi đã xác định một Tháng ví dụ như gõ để đối phó với vấn đề nhớ thứ tự tháng / ngày để tránh nhầm lẫn về việc liệu ngày 7 tháng 6 có được viết không {6,7} (Kiểu Mỹ) hoặc {7,6} (Phong cách châu Âu).**

**Tôi đã cân nhắc việc giới thiệu các loại riêng biệt Ngày và Năm để đối phó với sự nhầm lẫn có thể xảy ra**

**Ngày {1995, Month :: jul, 27} và Ngày {27, tháng :: jul, 1995}. Tuy nhiên, những loại này sẽ không hữu ích bằng**

**như là Tháng kiểu. Hầu như tất cả các lỗi như vậy đều được phát hiện tại thời điểm chạy - ngày 26 tháng 7 năm 27**



**472** **Các lớp học** **Chương 16**

**không phải là một ngày phổ biến trong công việc của tôi. Đối phó với các niên đại lịch sử trước năm 1800 hoặc lâu hơn là một vấn đề khó khăn tốt nhất nên dành cho các nhà sử học lão luyện. Hơn nữa, ngày trong tháng không thể được kiểm tra chính xác tách biệt với tháng và năm của nó.**

**Để giúp người dùng không phải đề cập rõ ràng đến năm và tháng ngay cả khi chúng được ngụ ý theo ngữ cảnh, tôi đã thêm một cơ chế cung cấp giá trị mặc định. Lưu ý rằng đối vớiTháng NS {} đưa ra giá trị (mặc định) 0 giống như đối với các số nguyên mặc dù nó không phải là một Tháng (§8.4). Tuy nhiên, trong trường hợp này, đó chính xác là những gì chúng tôi muốn: một giá trị bất hợp pháp khác để đại diện cho '' chọn giá trị mặc định. '' Cung cấp giá trị mặc định (ví dụ: giá trị mặc định choNgày đối tượng) là một vấn đề thiết kế phức tạp. Đối với một số loại, có một mặc định thông thường (ví dụ:0 cho số nguyên); đối với những người khác, không có mặc định nào có ý nghĩa; và cuối cùng, có một số loại (chẳng hạn nhưNgày) trong đó câu hỏi về việc có cung cấp giá trị mặc định hay không là không quan trọng. Trong những trường hợp như vậy, tốt nhất là - a**

**chủ yếu để có thể tháo gỡ**

**Tôi đã bỏ qua kỹ thuật bộ nhớ cache khỏi §16.2.9 vì không cần thiết đối với kiểu đơn giản này. Nếu cần, nó có thể được thêm vào dưới dạng chi tiết triển khai mà không ảnh hưởng đến giao diện người dùng.**

**Đây là một ví dụ nhỏ - và nguyên bản - về cách Ngàys có thể được sử dụng:**

**void f (Ngày & d)**

**{**

**Ngày lvb\_day {16, Month :: dec, d.year ()};**

**if (d.day () == 29 && d.month () == Tháng :: feb) {**

* ***...***

**}**

**if (nửa đêm ()) d.add\_day (1);**

**cout << "day sau:" << d + 1 << '\ n';**

**Ngày đ; //*khởi tạo thành ngày mặc định***



**cin >> đ;**

**if (dd == d) cout << "H**

**}**

**Điều này giả định rằng toán tử bổ sung, +, đã được khai báo cho NgàyNS. Tôi làm điều đó trong §16.3.3. Lưu ý việc sử dụng bằng cấp rõ ràng củaTháng mười hai và tháng Hai qua Tháng. Tôi đã sử dụng mộtlớp enum (§8.4.1) đặc biệt để có**

**thể sử dụng tên ngắn cho các tháng, nhưng cũng đảm bảo rằng việc sử dụng chúng sẽ không bị tối nghĩa hoặc mơ hồ.**

**Tại sao việc xác định một loại cụ thể cho một thứ đơn giản như ngày lại là điều cần thiết? Rốt cuộc, chúng ta chỉ có thể xác định một cấu trúc dữ liệu đơn giản:**

**struct Ngày {**

**int ngày, tháng, năm;**

**};**

**Mỗi lập trình viên sau đó có thể quyết định phải làm gì với nó. Tuy nhiên, nếu chúng tôi làm điều đó, mọi người dùng sẽ phải thao tác các thành phần củaNgàytrực tiếp hoặc cung cấp các chức năng riêng biệt để làm như vậy. Trên thực tế, khái niệm về ngày tháng sẽ nằm rải rác trong hệ thống, điều này sẽ làm cho nó khó hiểu, ghi chép hoặc thay đổi. Không thể tránh khỏi, cung cấp một khái niệm chỉ là một cấu trúc đơn giản**



**Mục 16.3** **Lớp bê tông** **473**

**gây ra công việc bổ sung cho mọi người dùng của cấu trúc.**

**Ngoài ra, mặc dù Ngày loại có vẻ đơn giản, cần phải suy nghĩ một chút để hiểu đúng. Ví dụ, tăng mộtNgày phải đối phó với năm nhuận, với thực tế là các tháng có độ dài khác nhau, v.v. Ngoài ra, đại diện ngày-tháng và năm khá kém đối với nhiều ứng dụng. Nếu chúng tôi quyết định thay đổi nó, chúng tôi sẽ chỉ cần sửa đổi một bộ chức năng được chỉ định. Ví dụ, để đại diện cho mộtNgày là số ngày trước hoặc sau ngày 1 tháng 1 năm 1970, chúng tôi chỉ cần thay đổi Ngàycủa các chức năng thành viên.**

**Để đơn giản hóa, tôi quyết định loại bỏ khái niệm thay đổi ngày mặc định. Làm như vậy sẽ loại bỏ một số cơ hội gây nhầm lẫn và khả năng xảy ra các điều kiện chủng tộc trong một chương trình đa luồng (§5.3.1). Tôi đã nghiêm túc xem xét việc loại bỏ hoàn toàn khái niệm về một ngày mặc định. Điều đó sẽ buộc người dùng phải nhất quán**

**và đáng ngạc nhiên, và hơn thế nữa tôi**

**struction (§17.3.3). Điều đó có nghĩa là tôi, với tư cách là người thiết kếNgày, phải chọn ngày mặc định. Tôi chọn ngày 1 tháng 1 năm 1970, vì đó là điểm bắt đầu cho các quy trình thời gian thư viện chuẩn C và C ++ (§35.2, §43.6). Rõ ràng là loại bỏset\_default\_date () gây ra một số mất tính tổng quát củaNgày. Tuy nhiên, thiết kế - bao gồm cả thiết kế lớp - là việc đưa ra quyết định, thay vì chỉ quyết định trì hoãn chúng hoặc để ngỏ tất cả các tùy chọn cho người dùng.**

**Để duy trì cơ hội cải tiến trong tương lai, tôi đã tuyên bố default\_date () như một chức năng trợ giúp:**

**const Ngày & Chrono :: default\_date ();**

**Điều đó không nói lên bất cứ điều gì về cách ngày mặc định thực sự được đặt.**

**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:**

**Ngày :: Ngày (int dd, Tháng mm, int yy)**



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

**{**

**nếu (y == 0) y = mặc định**

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

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

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

**}**

**Hàm tạo kiểm tra xem dữ liệu được cung cấp có biểu thị một Ngày. Nếu không, hãy nói, cho{30, Month :: feb, 1994}, nó ném ra một ngoại lệ (§2.4.3.1, Chương 13), cho biết rằng đã xảy ra sự cố. Nếu dữ liệu được cung cấp có thể chấp nhận được, thì quá trình khởi tạo rõ ràng đã được thực hiện. Khởi tạo là một hoạt động tương đối phức tạp vì nó liên quan đến việc xác nhận dữ liệu. Điều này là khá điển hình. Mặt khác, một khi Ngày đã được tạo, nó có thể được sử dụng và sao chép mà không cần kiểm tra thêm. Nói cách khác, hàm tạo thiết lập giá trị bất biến cho lớp (trong trường hợp này, nó biểu thị một ngày hợp lệ). Các chức năng thành viên khác có thể dựa vào sự bất biến đó và phải duy trì nó. Kỹ thuật thiết kế này có thể đơn giản hóa mã rất nhiều (xem §2.4.3.2, §13.4).**

**Tôi đang sử dụng giá trị Tháng{} - không đại diện cho một tháng và có giá trị số nguyên 0 - đại diện cho '' chọn tháng mặc định. '' Tôi có thể đã xác định một điều tra viên trong Tháng đặc biệt để**



**474** **Các lớp học** **Chương 16**

**đại diện cho điều đó. Nhưng tôi quyết định rằng tốt hơn nên sử dụng một giá trị rõ ràng là bất thường để đại diện cho '' chọn tháng mặc định '' hơn là đưa ra hình thức rằng có 13 tháng trong một năm. Lưu ý rằng Tháng{}, Ý nghĩa 0, có thể được sử dụng vì nó nằm trong phạm vi được đảm bảo cho phép liệt kêTháng (§8.4).**

**Tôi sử dụng cú pháp khởi tạo thành viên (§17.4) để khởi tạo các thành viên. Sau đó, tôi kiểm tra0**

**và sửa đổi các giá trị nếu cần. Điều này rõ ràng không mang lại hiệu suất tối ưu trong trường hợp (hy vọng là hiếm) xảy ra lỗi, nhưng việc sử dụng các trình khởi tạo thành viên để lại cấu trúc của mã rõ ràng. Điều này làm cho phong cách ít lỗi hơn và dễ bảo trì hơn so với các lựa chọn thay thế. Nếu tôi nhắm đến hiệu suất tối ưu, tôi đã sử dụng ba hàm tạo riêng biệt hơn là một hàm tạo duy nhất với các đối số mặc định.**

**Tôi đã xem xét việc làm cho mã**

**người dùng thứ này phức tạp hơn**

**điền vào khoảng trống (vectơ <Ngày> & aa) {**

**trong khi (cin) {**

**Ngày d;**

**cố gắng {**

**cin >> d;**

**}**

**bắt (Date :: Bad\_date) {**

* ***... xử lý lỗi của tôi ...***

**tiếp tục;**

**}**

**aa.push\_back (d);** **// *xem §4.4.2***

**}**

**}**

**Tuy nhiên, kiểm tra rằng một {d, m, y} tập hợp các giá trị là một ngày hợp lệ không phải là một phép tính phụ thuộc vào biểu diễn của một Ngày, vì vậy tôi đã triển khai is\_valid () về chức năng trợ giúp:**



**bool Ngày :: is\_valid ()**

**{**

**return is\_date (d, m, y);**

**}**

**Tại sao có cả hai is\_valid () và is\_date ()? Trong ví dụ đơn giản này, chúng tôi có thể quản lý chỉ với một, nhưng tôi có thể hình dung các hệ thống nơiis\_date () (như ở đây) kiểm tra rằng một (d, m, y)-tuple đại diện cho một ngày hợp lệ và ở đâu is\_valid () kiểm tra bổ sung xem liệu ngày đó có thể được trình bày một cách hợp lý hay không. Ví dụ,is\_valid () có thể từ chối các ngày từ trước khi lịch hiện đại được sử dụng phổ biến.**

**Như phổ biến đối với các loại bê tông đơn giản như vậy, các định nghĩa của NgàyCác chức năng của thành viên khác nhau giữa các chức năng tầm thường và không quá phức tạp. Ví dụ:**

**nội tuyến int Date :: day () const {**

**trả lại d;**

**}**



|  |  |  |
| --- | --- | --- |
| **Mục 16.3.1** | **Chức năng thành viên** | **475** |
| **Ngày & tháng :: add\_month (int n) {** |  |  |
| **if (n == 0) trả về ∗cái này;** |  |  |
| **nếu (n> 0) {** |  |  |
| **int delta\_y = n / 12;** | **// *số cả năm*// *số tháng trước*//** |  |
| **int mm = static\_cast <int> (m) + n% 12; nếu** | ***lưu ý: số thập phân được biểu*** |  |
| **(12 <mm) {** | ***thị bằng 12*** |  |

**+ + delta\_y;**

**mm - = 12;**

**}**

**// *... xử lý c***

**y + = delta\_y;**

**m = static\_cast <Tháng> (mm); trở**

**lại∗cái này;**

**}**

**// *... xử lý âm n ...***

**trở lại ∗cái này;**

**}**

**Tôi sẽ không gọi mã cho add\_month () khá.** **Trên thực tế, nếu tôi thêm tất cả các chi tiết, nó thậm chí có thể**

**tiếp cận mức độ phức tạp của mã thế giới thực tương đối đơn giản. Điều này chỉ ra một vấn đề: thêm một tháng là khái niệm đơn giản, vậy tại sao mã của chúng ta lại trở nên phức tạp? Trong trường hợp này, lý do làd, m, y việc biểu diễn không thuận tiện cho máy tính cũng như cho chúng ta. Một đại diện tốt hơn (cho nhiều mục đích) sẽ chỉ đơn giản là một số ngày kể từ '' ngày không '' được xác định (ví dụ: ngày 1 tháng 1 năm 1970). Điều đó sẽ làm cho tính toán trênNgàyđơn giản với chi phí phức tạp trong việc cung cấp đầu ra phù hợp cho con người**



**Lưu ý rằng phép gán a không cần hàm hủy vì Ngày không sở hữu tài nguyên và không yêu cầu dọn dẹp khi nó vượt ra khỏi phạm vi (§3.2.1.2).**

**16.3.2 Chức năng của người trợ giúp**

**Thông thường, một lớp có một số hàm được liên kết với nó mà không cần được định nghĩa trong chính lớp đó vì chúng không cần truy cập trực tiếp vào biểu diễn. Ví dụ:**

**int diff (Ngày a, Ngày b);** **// *số ngày trong phạm vi [a, b) hoặc [b, a)***

**bool is\_leapyear (int y);**

**bool is\_date (int d, Tháng m, int y);**

**const Ngày & default\_date ();**

**Date next\_weekday (Ngày d);**

**Date next\_saturday (Ngày d);**



**476** **Các lớp học** **Chương 16**

**Việc xác định các hàm như vậy trong bản thân lớp sẽ làm phức tạp giao diện lớp và tăng số lượng các hàm có khả năng cần được kiểm tra khi xem xét thay đổi biểu diễn.**

**Làm thế nào những chức năng như vậy '' liên kết '' với lớp Ngày? Trong C ++ đầu tiên, cũng như trong C, các khai báo của chúng đơn giản được đặt trong cùng một tệp như khai báo của lớpNgày. Người dùng cần Ngàys sẽ cung cấp tất cả chúng bằng cách bao gồm tệp xác định giao diện (§15.2.2). Ví dụ:**

**# bao gồm "Date.h"**

**Ngoài ra (hoặc cách khác), chúng ta có thể làm cho liên kết rõ ràng bằng cách bao bọc lớp và các chức năng trợ giúp của nó trong một không gian tên (§14.3.1):**

**không gian tên Chrono {**

**Ngày học {/ \* *...* \* /};**

**int diff (Ngày a, Ngày b);**

**bool is\_leapyear (int y);**

**bool is\_date (int d, Tháng m, int y);**

**const Ngày & default\_date ();**

**Date next\_weekday (Ngày d);**

**Date next\_saturday (Ngày**

**d); //*...***

**}**

**Các Chrono không gian tên đương nhiên cũng sẽ chứa các lớp liên quan, chẳng hạn như Thời gian và Đồng hồ bấm giờ và các chức năng trợ giúp của họ. Sử dụng không gian tên để chứa một lớp đơn lẻ thường là một sự phức tạp quá mức dẫn đến sự bất tiện.**

**Đương nhiên, hàm trợ giúp phải được định nghĩa ở đâu đó:**

**bool Chrono :: is\_date (int d, Month m, int y) {**



**int ndays;**

**chuyển đổi (m) {**

**trường hợp Tháng :: feb:**

**ndays = 28 + is\_leapyear (y);**

**nghỉ;**

**case Tháng :: apr: case Tháng :: jun: case Tháng :: sep: case Tháng :: nov:**

**thứ n = 30;**

**nghỉ;**

**case Tháng :: jan: case Tháng :: mar: case Tháng :: may: case Tháng :: jul: case**

**Tháng :: aug: case Tháng :: oct: case Tháng :: dec:**

**thứ n = 31;**

**nghỉ;**

**vỡ nợ:**

**trả về sai;**

**}**

**trả về 1 <= d && d <= ndays;**

**}**



**// *bình đẳng***

**Mục 16.3.2** **Chức năng của người trợ giúp** **477**

**Tôi đang cố tình hơi hoang tưởng ở đây. MỘTTháng không nên ở bên ngoài tháng một đến Tháng mười hai phạm vi, nhưng nó có thể (ai đó có thể đã cẩu thả với một diễn viên), vì vậy tôi kiểm tra.**

**Rắc rối default\_date finally trở thành:**

**const Ngày & Chrono :: default\_date () {**

**static Date d {1, Month :: jan, 1970};**

**trả lại d;**

**}**

**16.3.3 Op bị quá tải**

**Thường hữu ích khi thêm f định nghĩa toán tử bình**

**đẳng, ==, làm việc cho NgàyNS:**

**toán tử bool nội tuyến == (Ngày a, Ngày b) {**

**return a.day () == b.day () && a.month () == b.month () && a.year () == b.year ();**

**}**

**Các ứng cử viên rõ ràng khác là:**

**toán tử bool! = (Ngày tháng);** **// *bất bình đẳng***

**toán tử bool <(Ngày tháng);** **// *ít hơn***

**toán tử bool> (Ngày tháng); //** **// *lớn hơn***

***...***

**Ngày & toán tử ++ (Ngày & d) {return d.add\_day (1); }** **// *tăng Ngày lên một ngày*//**

**Ngày & toán tử −− (Ngày & d) {return d.add\_day (−1); }** ***giảm ngày một ngày***

**Ngày & toán tử + = (Ngày & d, int n) {return d.add\_day (n); }** **// *thêm n ngày***

****

**Ngày & toán tử - = (Ngày &**

**Toán tử ngày + (Ngày d, trong**

**Toán tử ngày− (Ngày d, int n) {return d + = n; }** **// *trừ n ngày***

**ostream & operator << (ostream &, Date d);** **// *đầu ra d***

**istream & operator >> (istream &, Date & d);** **// *đọc thành d***

**Các toán tử này được định nghĩa trong Chrono cùng với Ngày để tránh các vấn đề quá tải và hưởng lợi từ tra cứu phụ thuộc vào đối số (§14.2.4).**

**Vì Ngày, những nhà khai thác này có thể được coi là những tiện ích đơn thuần. Tuy nhiên, đối với nhiều loại - chẳng hạn như**

**số phức (§18.3), vectơ (§4.4.1) và các đối tượng giống hàm (§3.4.3, §19.2.2) - việc sử dụng các toán tử thông thường đã cố định trong tâm trí mọi người đến mức định nghĩa của chúng gần như là bắt buộc . Quá tải toán tử được thảo luận trong Chương 18.**

**Vì Ngày, Tôi đã bị cám dỗ để cung cấp + = và - = như các chức năng thành viên thay vì add\_day (). Tôi đã**

**làm như vậy, tôi sẽ tuân theo một thành ngữ phổ biến (§3.2.1.1).**

**Lưu ý rằng việc gán và khởi tạo sao chép được cung cấp theo mặc định (§16.3, §17.3.3).**

****

**478** **Các lớp học** **Chương 16**

**16.3.4 Tầm quan trọng của các lớp bê tông**

**Tôi gọi các loại đơn giản do người dùng xác định, chẳng hạn như Ngày, *các loại bê tông* để phân biệt chúng với các lớp trừu tượng (§3.2.2) và phân cấp lớp (§20.4), và cũng để nhấn mạnh sự giống nhau của chúng với các kiểu tích hợp như NS và char. Các lớp bê tông được sử dụng giống như các loại tích hợp sẵn. Các loại bê tông còn được gọi là*các loại giá trị* và việc sử dụng chúng *lập trình hướng giá trị*. Mô hình sử dụng của chúng và '' triết lý '' đằng sau thiết kế của chúng hoàn toàn khác với những gì thường được gọi là lập trình hướng đối tượng (§3.2.4, Chương 21).**

**Mục đích của loại bê tông là làm tốt và hiệu quả một việc duy nhất, tương đối đơn giản. Nó thường không nhằm mục đích cung cấp cho người dùng các phương tiện để sửa đổi hành vi của một loại cụ thể. Đặc biệt, các loại bê tông a**

**§20.3.2).**

**Nếu bạn không thích một số chi tiết của một kiểu cụ thể, bạn xây dựng một kiểu mới với hành vi mong muốn. Nếu bạn muốn '' tái sử dụng '' một kiểu cụ thể, bạn sử dụng nó trong việc triển khai kiểu mới chính xác như bạn đã sử dụngNS. Ví dụ:**

**lớp Date\_and\_time {**

**riêng:**

**Ngày d;**

**Thời gian t;**

**công cộng:**

**Date\_and\_time (Ngày d, Thời gian t); Date\_and\_time (int d,**

**Date :: Tháng m, int y, Time t); //*...***

**};**

**Ngoài ra, cơ chế lớp dẫn xuất được thảo luận trong Chương 20 có thể được sử dụng để định nghĩa các kiểu mới từ một lớp cụ thể bằng cách mô tả sự khác biệt mong muốn. Định nghĩa củaVec từ vectơ (§4.4.1.2) là một ví dụ về điều này. Tuy nhiên, việc dẫn xuất từ một lớp cụ thể nên được thực hiện cẩn thận và hiếm khi xảy ra vì thiếu các chức năng ảo và thông tin kiểu thời gian chạy (§17.5.1.4, Chương 22).**



**Với một hợp lý tốt**

**thời gian hoặc không gian. Đặc biệt, không cần chuyển hướng thông qua con trỏ để truy cập vào các đối tượng của các lớp cụ thể và không có dữ liệu '' quản lý nhà '' nào được lưu trữ trong các đối tượng của các lớp cụ thể. Kích thước của một loại cụ thể được biết trước tại thời điểm biên dịch để các đối tượng có thể được phân bổ trên ngăn xếp thời gian chạy (nghĩa là không có hoạt động lưu trữ miễn phí). Bố cục của một đối tượng được biết trước tại thời điểm biên dịch để nội tuyến của các hoạt động đạt được một cách đáng kể. Tương tự, khả năng tương thích bố cục với các ngôn ngữ khác, chẳng hạn như C và Fortran, không cần nỗ lực đặc biệt.**

**Một tập hợp tốt các loại như vậy có thể cung cấp nền tảng cho các ứng dụng. Đặc biệt, chúng có thể được sử dụng để làm cho các giao diện cụ thể hơn và ít bị lỗi hơn. Ví dụ:**

**Tháng do\_sự gì đó (Ngày d);**

**Điều này ít có khả năng bị hiểu lầm hoặc sử dụng sai hơn nhiều so với:**

**int do\_something (int d);**

**Thiếu kiểu cụ thể có thể dẫn đến các chương trình tối nghĩa và lãng phí thời gian khi mỗi lập trình viên viết mã để thao tác trực tiếp các cấu trúc dữ liệu '' đơn giản và được sử dụng thường xuyên '' được biểu diễn dưới dạng**



**Mục 16.3.4** **Tầm quan trọng của các lớp bê tông** **479**

**các tập hợp đơn giản của các loại dựng sẵn. Ngoài ra, việc thiếu '' các loại hiệu quả nhỏ '' phù hợp trong một ứng dụng có thể dẫn đến sự kém hiệu quả về thời gian chạy và không gian khi sử dụng các lớp quá chung chung và đắt tiền.**

**16.4 Lời khuyên**

1. **Biểu diễn các khái niệm dưới dạng các lớp; §16.1.**
2. **Tách giao diện của một lớp khỏi việc triển khai nó; §16.1.**
3. **Sử dụng dữ liệu công khai (cấu trúcs) chỉ khi nó thực sự chỉ là dữ liệu và không có bất biến nào có ý nghĩa đối với các**

**thành viên dữ liệu; §16.2.**

1. **Xác định hàm tạo t**
2. **Theo mặc định, khai báo các hàm tạo đối số đơn rõ ràng; §16.2.6.**
3. **Khai báo một hàm thành viên không sửa đổi trạng thái của đối tượng của nó hăng sô; §16.2.9.**
4. **Loại bê tông là loại đơn giản nhất của lớp. Nếu có thể, hãy thích một kiểu cụ thể hơn các lớp phức tạp hơn và các cấu trúc dữ liệu đơn giản; §16.3.**
5. **Chỉ biến một hàm thành thành viên nếu nó cần truy cập trực tiếp vào biểu diễn của một lớp; §16.3.2.**
6. **Sử dụng một không gian tên để làm cho mối liên kết giữa một lớp và các chức năng trợ giúp của nó trở nên rõ ràng;**

**§16.3.2.**

1. **Tạo một hàm thành viên không thay đổi giá trị của đối tượng của nó a hăng sô chức năng thành viên; §16.2.9.1.**
2. **Tạo một hàm cần truy cập vào biểu diễn của một lớp nhưng không cần được gọi cho một đối tượng cụ thể a tĩnh chức năng thành viên; §16.2.12.**



***Trang này cố ý để trống***



**17**



**Xây dựng**

***Sự thiếu hiểu biết thường xuyên dẫn đến sự tự tin***

***hơn là kiến thức.***

***- Charles Darwin***

* **Giới thiệu**
* **Người xây dựng và người phá hủy**

**Người xây dựng và Người bất biến; Kẻ hủy diệt và Tài nguyên; Cơ sở và Thành viên Hủy bỏ; Gọi các cấu tạo và hủy cấu trúc;ảo Kẻ hủy diệt**

**• Khởi tạo đối tượng lớp**

**Khởi tạo không có hàm tạo; Khởi tạo bằng Constructors; Trình xây dựng mặc định; Trình tạo danh sách khởi tạo**

* **Thành viên và Cơ sở Khởi tạo Thành viên Khởi tạo**



**Thành viên Initializatio**

**• Sao chép và di chuyển**

**Sao chép; Di chuyển**

**• Tạo hoạt động mặc định**

**Mặc định rõ ràng; Hoạt động mặc định; Sử dụng Hoạt động Mặc định;xóa bỏd Chức năng**

**• Lời khuyên**

**17.1 Giới thiệu**

**Chương này tập trung vào các khía cạnh kỹ thuật của '' vòng đời '' của một đối tượng: Làm thế nào để chúng ta tạo một đối tượng, chúng ta sao chép nó như thế nào, chúng ta di chuyển nó ra sao và làm thế nào để chúng ta dọn dẹp nó sau khi nó biến mất? Định nghĩa thích hợp của '' sao chép '' và '' di chuyển '' là gì? Ví dụ:**



**482**

**Xây dựng, Dọn dẹp, Sao chép và Di chuyển**

**Chương 17**

**chuỗi nhận dạng (string arg) {**

**// *chuỗi được truyền bởi giá trị (được sao chép vào arg)***

**trở lại đối số;**

**// *trả về chuỗi (di chuyển giá trị của arg ra khỏi danh tính () đến một người gọi)***

**}**

**int main ()**

**{**

**string s1 {"Adams"};** **// *khởi tạo chuỗi (cấu trúc trong s1).*// *sao chép***

**s1 = indet (s1);** ***s1 vào danh tính ()*// *di chuyển kết quả của bản***

***sắc (s1) vào s1;***

**chuỗi s2 {"Pratchett**

**s1 = s2;**

* ***cả s1 và s2 đều có giá trị "Pratchett".***

**}**

**Rõ ràng, sau cuộc gọi của nhận dạng (), giá trị của s1 phải là "Adams ". Chúng tôi sao chép giá trị củas1 vào cuộc tranh luận tranh luận, sau đó chúng tôi di chuyển giá trị của tranh luận ra khỏi lệnh gọi hàm và (quay lại) vào s1. Tiếp theo, chúng tôi xây dựngs2 với giá trị "Prachett " và sao chép nó vào s1. Cuối cùng, ở lối ra từchủ chốt() chúng tôi phá hủy các biến s1 và s2. Sự khác biệt giữa *di chuyển* và *sao chép* là sau một lần sao chép, hai đối tượng phải có cùng giá trị, trong khi sau một lần di chuyển, nguồn của lần di chuyển không bắt buộc phải có giá trị ban đầu. Các chuyển động có thể được sử dụng khi đối tượng nguồn sẽ không được sử dụng nữa. Chúng đặc biệt hữu ích để thực hiện khái niệm di chuyển tài nguyên (§3.2.1.2, §5.2).**

**Một số chức năng được sử dụng ở đây:**

* **Một phương thức khởi tạo khởi tạo một dây với một chuỗi ký tự (được sử dụng cho s1 và s2)**
* **Một phương thức tạo bản sao sao chép một dây (vào đối số hàm tranh luận)**
* **Một hàm tạo di chuyển di chuyển giá trị của một dây (từ tranh luận ra khỏi nhận dạng () thành một biến tạm thời giữ kết quả là nhận dạng (s1))**
* **Một phép gán di chuyển di chuyển giá trị của một dây (từ biến tạm thời giữ kết quả là nhận**



**dạng (s1) NS**

* **Một bản phân công c**
* **Một trình hủy giải phóng các tài nguyên thuộc sở hữu của s1, s2và biến tạm thời giữ**

**kết quả của nhận dạng (s1)**

**Một trình tối ưu hóa có thể loại bỏ một số công việc này. Ví dụ, trong ví dụ đơn giản này, biến tạm thời thường bị loại bỏ. Tuy nhiên, về nguyên tắc, các thao tác này được thực thi.**

**Bộ tạo, sao chép và di chuyển các hoạt động gán và bộ hủy hỗ trợ trực tiếp chế độ xem quản lý tài nguyên và thời gian tồn tại. Một đối tượng được coi là một đối tượng thuộc kiểu của nó sau khi phương thức khởi tạo của nó hoàn thành và nó vẫn là một đối tượng thuộc kiểu của nó cho đến khi trình hủy của nó bắt đầu thực thi. Sự tương tác giữa thời gian tồn tại của đối tượng và các lỗi được khám phá thêm trong §13.2 và §13.3. Đặc biệt, chương này không thảo luận về vấn đề của các đối tượng được xây dựng một nửa và một nửa bị phá hủy.**

**Xây dựng các đối tượng đóng một vai trò quan trọng trong nhiều thiết kế. Việc sử dụng đa dạng này được phản ánh trong phạm vi và tính linh hoạt của các tính năng ngôn ngữ hỗ trợ khởi tạo.**

**Các hàm tạo, hàm hủy và các thao tác sao chép và di chuyển cho một kiểu không tách biệt nhau về mặt logic. Chúng ta phải xác định chúng như một tập hợp phù hợp hoặc gặp phải các vấn đề về hiệu suất hoặc logic. Nếu một lớp họcNS có một trình hủy thực hiện một nhiệm vụ quan trọng, chẳng hạn như phân bổ giao dịch trong cửa hàng miễn phí hoặc phát hành khóa, lớp có thể cần bổ sung đầy đủ các chức năng:**



|  |  |  |
| --- | --- | --- |
| **Mục 17.1** | **Giới thiệu** | **483** |
| **lớp X {** |  |  |
| **X (Kiểu nào đó);** | **// *''hàm tạo thông thường '': tạo một đối tượng*//** |  |
| **NS();** | ***nhà xây dựng mặc định*** |  |
| **X (const X &);** | **// *sao chép hàm tạo*** |  |
| **X (X &&);** | **// *di chuyển hàm tạo*** |  |
| **X & operator = (const X &); //*sao chép phân công: dọn dẹp mục tiêu và sao chép*X** | |  |
| **& operator = (X &&);** | **// *chuyển nhiệm vụ: dọn dẹp mục tiêu và di*** |  |
| **NS();** | ***chuyển*// *hủy: dọn dẹp*** |  |

**// *...***

**};**

**Có năm tình huống trong w**

* **Là nguồn gốc của một**
* **Là một bộ khởi tạo đối tượng**
* **Là một đối số của hàm**
* **Là một hàm trả về giá trị**
* **Như một ngoại lệ**

**Trong mọi trường hợp, hàm tạo sao chép hoặc di chuyển sẽ được áp dụng (trừ khi nó có thể được tối ưu hóa đi).**

**Ngoài việc khởi tạo các đối tượng và đối tượng được đặt tên trên cửa hàng miễn phí, các hàm tạo được sử dụng để khởi tạo các đối tượng tạm thời (§6.4.2) và thực hiện chuyển đổi kiểu rõ ràng (§11.5).**

**Ngoại trừ hàm tạo thông thường, các hàm thành viên đặc biệt này có thể được tạo bởi trình biên dịch; xem §17.6.**

**Chương này có đầy đủ các quy tắc và kỹ thuật. Những điều đó là cần thiết để hiểu đầy đủ, nhưng hầu hết mọi người chỉ học các quy tắc chung từ các ví dụ.**

**17.2 Cấu tạo và chất hủy**

**Chúng ta có thể chỉ định cách khởi tạo một đối tượng của một lớp bằng cách định nghĩa một phương thức khởi tạo (§16.2.5, §17.3). Để bổ sung cho con**



**phá hủy một đối tượng (ví dụ:**

**quản lý tài nguyên trong C ++ dựa trên các cặp hàm tạo / hủy. Vì vậy, các kỹ thuật khác dựa trên một cặp hành động, chẳng hạn như thực hiện / hoàn tác, bắt đầu / dừng, trước / sau, v.v. Ví dụ:**

**struct Tracer {**

**chuỗi lộn xộn;**

**Tracer (const string & s): mess {s} {clog << mess; }**

**˜Tracer () {clog << "˜" << mess; }**

**};**

**void f (const vector <int> & v) {**

**Tracer tr {"in f () \ n"};**

**for (auto x: v) {**

**Tracer tr {string {"v loop"} + to <string> (x) + '\ n'}; //*§25.2.5.1*//**

***...***

**}**

**}**



**484** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Chúng tôi có thể thử một cuộc gọi:**

**f ({2,3,5});**

**Điều này sẽ in ra luồng ghi nhật ký:**

**in\_f ()**

**v vòng lặp 2**

**˜v vòng lặp 2**

**v vòng lặp 3**

**˜v vòng lặp 3**

**v vòng lặp 5**

**˜v vòng lặp 5**

**˜in\_f ()**

**17.2.1 Cấu trúc và Bất biến**

**Một thành viên có cùng tên với lớp của nó được gọi là *constructor*. Ví dụ:**

**Vectơ lớp {**

**công cộng:**

**Véc tơ (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 {**

**NS();** **// *fine***

**void S (int);** **// *lỗi: không có loại nào có thể được chỉ định cho một phương thức***

**int S;** ***khởi tạo*// *error: tên lớp phải biểu thị một phương thức khởi tạo***



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

**};**

**Công việc của một hàm tạo là khởi tạo một đối tượng của lớp nó. Thông thường, việc khởi tạo đó phải thiết lập*lớp bất biến*, nghĩa là, một cái gì đó phải giữ bất cứ khi nào một hàm thành viên được gọi (từ bên ngoài lớp). Xem xét:**

**Vectơ lớp {**

**công cộng:**

**Véc tơ (int s);**

* ***...***

**riêng:**

**kép∗ độ cao; //*elem trỏ đến một mảng nhân đôi sz*int sz; //*sz không âm***

**};**

* **đây (như thường lệ), bất biến được phát biểu dưới dạng nhận xét: ''elem trỏ đến một mảng sz nhân đôi '' và ''sz là không âm. '' Hàm tạo phải làm cho điều đó đúng. Ví dụ:**



**Mục 17.2.1** **Người xây dựng và Người bất biến** **485**

**Vectơ :: Vectơ (int s)**

**{**

**if (s <0) ném Bad\_size {s}; sz**

**= s;**

**elem = new double [s];**

**}**

**Hàm tạo này cố gắng thiết lập bất biến và nếu không thể, nó sẽ ném ra một ngoại lệ. Nếu phương thức khởi tạo không thể thiết lập bất biến, không có đối tượng nào được tạo ra và phương thức khởi tạo phải đảm bảo rằng không có tài nguyên nào bị rò rỉ (§5.2, §13.3). Tài nguyên là bất cứ thứ gì chúng ta cần để có được và cuối cùng (rõ ràng hoặc ẩn ý) gi**

**là bộ nhớ (§3.2.1.2), khóa**

**Tại sao bạn lại xác định một**

* **Để tập trung nỗ lực thiết kế cho lớp (§2.4.3.2)**
* **Để làm rõ hành vi của lớp (ví dụ: trong các điều kiện lỗi; §13.2)**
* **Để đơn giản hóa định nghĩa của các hàm thành viên (§2.4.3.2, §16.3.1)**
* **Để làm rõ việc quản lý tài nguyên của lớp (§13.3)**
* **Để đơn giản hóa tài liệu của lớp**

**Trung bình, nỗ lực xác định một bất biến cuối cùng sẽ tiết kiệm được công việc.**

**17.2.2 Kẻ hủy diệt 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 có được 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. Chắc chắn, một hàm như vậy được gọi là*người phá hủy*. Tên của một trình hủy là˜ theo sau là tên lớp, chẳng hạn ˜Véc tơ (). Một nghĩa là làm mờ các hàm tạo của nó. Một cấu trúc d. Kẻ hủy diệt là ca**



**trên cửa hàng miễn phí bị xóa, v.v. Chỉ trong những trường hợp rất hiếm, người dùng mới cần gọi hàm hủy một cách rõ ràng (§17.2.4).**

**Kẻ hủy diệt thường dọn dẹp và giải phóng tài nguyên. Ví dụ:**

**Vectơ lớp {**

**công cộng:**

**Vectơ (int s): elem {new double [s]}, sz {s} {};** **// *hàm tạo: thu được bộ nhớ*//**

**˜Vector () {xóa [] elem; }** ***hủy: giải phóng bộ nhớ***

**// *...***

**riêng:**

**kép∗ độ cao; //*elem trỏ đến một mảng nhân đôi sz*int**

**sz; //*sz không âm***

**};**

**Ví dụ:**



**486** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Véc tơ∗ f (int s) {**

**Vectơ v1 (s);**

* ***...***

**trả về Vector mới (s + s);**

**}**

**void g (int ss)**

**{**

**Véc tơ∗ p = f**

**(ss); //*...***

**xóa p;**

**}**

**Đây, Vectơ v1 bị phá hủy khi thoát ra khỏi NS(). Ngoài ra,Véc tơ được tạo trên cửa hàng miễn phí bởi NS()sử dụng Mới bị phá hủy bởi cuộc gọi của xóa bỏ. Trong cả hai trường hợp,Véc tơHàm hủy của được gọi để giải phóng (deallocate) bộ nhớ được cấp phát bởi hàm tạo.**

**Điều gì sẽ xảy ra nếu hàm tạo không có đủ bộ nhớ? Ví dụ,NS∗sizeof (gấp đôi) hoặc**

**(s + s)∗sizeof (gấp đôi) có thể lớn hơn dung lượng bộ nhớ khả dụng (tính bằng byte). Trong trường hợp đó, một ngoại lệstd :: bad\_alloc (§11.2.3) được ném bởi Mới và cơ chế xử lý ngoại lệ gọi các hàm hủy thích hợp để tất cả bộ nhớ đã được lấy (và chỉ bộ nhớ đó) được giải phóng (§13.5.1).**

**Phong cách quản lý tài nguyên dựa trên hàm tạo / hàm hủy này được gọi là *Thu thập tài nguyên là khởi* *tạo* hoặc đơn giản *RAII* (§5.2, §13.3).**

**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 ++. Vùng chứa thư viện tiêu chuẩn, chẳng hạn nhưvectơ và sơ đồ không có thứ tự, sử dụng các biến thể của kỹ thuật này để cung cấp bộ nhớ cho các phần tử của chúng.**

**Một kiểu không có trình hủy được khai báo, chẳng hạn như kiểu tích hợp, được coi là có trình hủy không làm gì cả.**



**Một lập trình viên bị sao**

**chép hoặc di chuyển (§17.6)**

**17.2.3 Chất phá hủy cơ sở và thành viên**

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

1. **đầu tiên, hàm tạo gọi các hàm tạo lớp cơ sở của nó,**
2. **sau đó, nó gọi các hàm tạo thành viên, và**
3. **cuối cùng, nó tự thực thi cơ thể của mình.**

**Một trình hủy '' xé nhỏ '' một đối tượng theo thứ tự ngược lại:**

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

**Đặc biệt, một ảo căn cứ được xây dựng trước bất kỳ căn cứ nào có thể sử dụng nó và bị phá hủy sau tất cả những căn cứ đó (§21.3.5.1). Thứ tự này đảm bảo rằng một cơ sở hoặc một thành viên không được sử dụng trước khi nó được khởi tạo hoặc được sử dụng sau khi nó đã bị phá hủy. Lập trình viên có thể đánh bại điều này đơn giản và**



**Mục 17.2.3** **Cơ sở và phần tử hủy thành viên** **487**

**quy tắc thiết yếu, nhưng chỉ thông qua việc cố ý phá vỡ liên quan đến việc chuyển con trỏ đến các biến chưa được khởi tạo làm đối số. Làm như vậy vi phạm các quy tắc ngôn ngữ và kết quả thường là thảm hại.**

**Các hàm tạo thực thi các hàm khởi tạo thành viên và cơ sở theo thứ tự khai báo (không phải thứ tự của các trình khởi tạo): nếu hai hàm tạo sử dụng một thứ tự khác nhau, bộ hủy không thể (không có chi phí nghiêm trọng) đảm bảo sẽ hủy theo thứ tự ngược lại của cấu trúc. Xem thêm §17.4.**

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

**struct S1 {**

**dây;**

**};**

**S1 x;** **// *OK: xs là starta***

**Tương tự, khởi tạo thành viên có thể được sử dụng nếu cần các trình khởi tạo. Ví dụ:**

**struct X {X (int); };**

**struct S2 {**

**X x;**

**};**

**S2 x1;** **// *lỗi:***

**S2 x2 {1}; //*OK: x2.x được khởi tạo bằng 1***

**Xem thêm §17.3.1.**

**17.2.4 Gọi hàm tạo và hủy**

**Hàm hủy được gọi ngầm khi thoát khỏi phạm vi hoặc bằng xóa bỏ. 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 gặp (nhưng quan trọng) w**



**(như std :: vector) duy trì**

**push\_back () và pop\_back ()**

**cho một địa chỉ cụ thể:**

**void C :: push\_back (const X & a) {**

* ***...***

**new (p) X {a};** **// *sao chép cấu trúc một X với giá trị a trong địa chỉ p***

* ***...***

**}**

**Việc sử dụng hàm tạo này được gọi là '' vị trí Mới'' (§11.2.4).**

**Ngược lại, khi chúng ta xóa một phần tử, vùng chứa cần gọi hàm hủy của nó:**

**void C :: pop\_back ()**

**{**

* ***...***

**p−> ˜X (); //*hủy X trong địa chỉ p***

**}**



**488** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Các p−> ˜X () ký hiệu gọi NStrình hủy của ∗P. Ký hiệu đó không bao giờ được sử dụng cho một đối tượng bị phá hủy theo cách thông thường (bởi đối tượng của nó đi ra ngoài phạm vi hoặc đangxóa bỏNS).**

**Để có ví dụ đầy đủ hơn về quản lý rõ ràng các đối tượng trong vùng bộ nhớ, hãy xem §13.6.1. Nếu được khai báo cho một lớpNS, một hàm hủy sẽ được gọi ngầm bất cứ khi nào NS đi ra ngoài phạm vi hoặc là xóa bỏNS.**

**Điều này ngụ ý rằng chúng ta có thể ngăn chặn việc phá hủy mộtNS bằng cách khai báo hàm hủy của nó**

* **xóa (§17.6.4) hoặc riêng.**

**Trong số hai lựa chọn thay thế, sử dụng riêng là linh hoạt hơn. Ví dụ: chúng ta có thể tạo một lớp mà các đối tượng có thể bị hủy một cách rõ ràng, nhưng không hoàn toàn:**

**lớp Nonlocal {**

**công cộng:**

* ***...***

**void tiêu diệt ()**

**{thisprivate:**

**// *...***

**˜Nonlocal ();**

**// *đừng phá hủy một cách ngầm***

**};**

**void user ()**

**{**

**Phi địa phương x;**

**NS∗ p = new**

**Nonlocal; //*...***

**// *lỗi: không thể hủy bỏ một Nonlocal*//**

***VÂNG***

**xóa p;**

**p.destroy ();**

**// *lỗi: không thể hủy bỏ một Nonlocal*//**

***VÂNG***

**}**

**17.2.5 ảo Kẻ hủy diệt**

**Một hàm hủy có thể được khai báo là ảo, và thường phải dành cho một lớp có chức năng ảo. Ví dụ:**



**Hình dạng lớp {**

**công cộng:**

* ***...***

**ảo void draw () = 0; ảo**

**˜Shape ();**

**};**

**vòng tròn lớp {**

**công cộng:**

* ***...***

**void draw ();**

**khoanh tròn();** **// *ghi đè ˜Shape ()***

* ***...***

**};**

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



**Mục 17.2.5** **ảo Kẻ hủy diệt** **489**

**người dùng vô hiệu (Hình dạng∗ P) {**

**p−> draw ();** **// *gọi lệnh draw thích hợp ()***

* ***...***

**xóa p;** **// *gọi hàm hủy thích hợp***

**};**

**Có Hình dạngtrình hủy của không được ảo điều đó xóa bỏ sẽ không thể gọi hàm hủy lớp dẫn xuất thích hợp (ví dụ: NSircle ()). Thất bại đó sẽ làm cho các tài nguyên thuộc sở hữu của đối tượng bị xóa (nếu có) bị rò rỉ.**

**17.3 Đối tượng lớp I**

**Phần này thảo luận về cách khởi tạo các đối tượng của một lớp có và không có hàm tạo. Nó cũng chỉ ra cách xác định các hàm tạo để chấp nhận danh sách bộ khởi tạo đồng nhất có kích thước tùy ý (chẳng hạn như**

**{1,2,3} và {1,2,3,4,5,6}).**

**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:**

**int a {1};**

**char∗ p {nullptr};**

**Tương tự, chúng ta có thể khởi tạo các đối tượng của một lớp mà chúng ta chưa xác định hàm tạo bằng cách sử dụng**

* **khởi tạo thành viên,**
* **sao chép khởi tạo, hoặc**
* **khởi tạo mặc định (không có bộ khởi tạo hoặc có danh sách bộ khởi tạo trống).**



**Ví dụ:**

**struct Work {**

**tác giả chuỗi;**

**tên chuỗi;**

**int năm;**

**};**

**Work s9 {"Beethoven",**

**"Giao hưởng số 9 ở giọng thứ, Op. 125; Hợp xướng",**

**1824**

**};** **// *khởi tạo thành viên***

**Làm việc hiện\_trên {s9};** **// *sao chép khởi tạo***

**Không làm việc {};** **// *khởi tạo mặc định***

**Ba thành viên của hiện đang chơi là bản sao của s9. Khởi tạo mặc định của việc sử dụng{} được định nghĩa là khởi tạo của từng thành viên bởi {}. Vì thế,không ai**

**được khởi tạo thành {{}, {}, {}}, đó là {"", "",0} (§17.3.3).**



**490** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Trường hợp không có hàm tạo nào yêu cầu đối số được khai báo, thì cũng có thể loại bỏ hoàn toàn bộ khởi tạo. Ví dụ:**

**Làm việc alpha;**

**void f ()**

**{**

**Làm việc beta;**

* ***...***

**}**

**Đối với điều này, các quy tắc không rõ ràng như chúng ta có thể muốn. Đối với các đối tượng được cấp phát tĩnh (§6.4.2), các quy tắc chính xác như thể bạn h**

**và các đối tượng lưu trữ miễn phí, d**

**bers của loại tích hợp không được khởi tạo, vì vậy giá trị của bản beta Là {"", "",không xác định}.**

**Lý do cho biến chứng này là để cải thiện hiệu suất trong những trường hợp nguy cấp hiếm gặp. Ví dụ:**

**struct Buf {**

**số int;**

**char buf [16∗1024];**

**};**

**Bạn có thể sử dụng một Buf như một biến cục bộ mà không cần khởi tạo nó trước khi sử dụng nó làm mục tiêu cho một thao tác nhập. Hầu hết các khởi tạo biến cục bộ không quan trọng về hiệu suất và các biến cục bộ chưa được khởi tạo là một nguồn lỗi chính. Nếu bạn muốn khởi tạo được đảm bảo hoặc đơn giản là không thích bất ngờ, hãy cung cấp một trình khởi tạo, chẳng hạn như{}. Ví dụ:**

**Buf buf0;**

**// *được cấp phát tĩnh, do đó được khởi tạo theo mặc định***

**void f ()**

**{**

**Buf buf1;**

**// *để lại các yếu tố chưa được khởi tạo***



**Buf buf2 {};**

**NS∗ p1 = int mới; NS∗ p2// *\*p1*=new*chưa được khởi***

**int {}; NS∗ p3 = new int*tạo*{7};// *\*p2*//*\*p3== 0***

***== 7*// *...***

**}**

**Đương nhiên, khởi tạo thành viên chỉ hoạt động nếu chúng ta có thể truy cập các thành viên. Ví dụ:**

**mẫu <lớp T>**

**class Checked\_pointer {// *kiểm soát quyền truy cập thành viên T \**công**

**cộng:**

**T & nhà điều hành∗();** **// *kiểm tra nullptr và giá trị trả về***

* ***...***

**};**

**Checked\_pointer <int> p {new int {7}};** **// *lỗi: không thể truy cập pp***

**Nếu một lớp có một không riêng tưtĩnh thành viên dữ liệu, nó cần một phương thức khởi tạo để khởi tạo nó.**



**Mục 17.3.2** **Khởi tạo bằng cách sử dụng hàm tạo** **491**

**17.3.2 Khởi tạo bằng cách sử dụng hàm tạo**

**Trong trường hợp bản sao thành viên là không đủ hoặc không mong muốn, một phương thức khởi tạo có thể được xác định để khởi tạo một đối tượng. Đặc biệt, một phương thức khởi tạo thường được sử dụng để thiết lập một bất biến cho lớp của nó và thu được các tài nguyên cần thiết để thực hiện điều đó (§17.2.1).**

**Nếu một hàm tạo được khai báo cho một lớp, một số hàm tạo sẽ được sử dụng cho mọi đối tượng. Đó là một lỗi khi cố gắng tạo một đối tượng mà không có bộ khởi tạo thích hợp theo yêu cầu của các hàm tạo. thí dụ: Vì**

**struct X {**

**X (int);**

**};**

|  |  |  |
| --- | --- | --- |
| **X x0;** | | **// *lỗi: không có trình khởi tạo*** |
| **X x1 {};** | | **// *lỗi: bộ khởi tạo trống*//** |
| **X x2** | **{2};** | ***VÂNG*** |
| **X x3** | **{"hai"};** | **// *lỗi: loại bộ khởi tạo sai*// *lỗi: sai số bộ khởi tạo*// *OK:*** |
| **X x4** | **{1,2};** | ***một phương thức khởi tạo sao chép được định nghĩa*** |
| **X x5** | **{x4};** | ***ngầm (§17.6)*** |

**Lưu ý rằng hàm tạo mặc định (§17.3.3) sẽ biến mất khi bạn xác định một hàm tạo yêu cầu đối số; sau tất cả,X (int) nói rằng một NS được yêu cầu để xây dựng một NS. Tuy nhiên, hàm tạo bản sao không biến mất (§17.3.3); giả định là một đối tượng có thể được sao chép (một khi được xây dựng đúng cách). Trong trường hợp sau này có thể gây ra sự cố (§3.3.1), bạn có thể đặc biệt không cho phép sao chép (§17.6.4).**

**Tôi đã sử dụng {} ký hiệu để làm rõ sự thật rằng tôi đang khởi tạo. Tôi không (chỉ) gán một giá trị, gọi một hàm hoặc khai báo một hàm. Các{} ký hiệu để khởi tạo có thể được sử dụng để cung cấp các đối số cho một phương thức khởi tạo ở bất cứ nơi nào một đối tượng có thể được xây dựng. Ví dụ:**

**struct Y: X {**

**X m {0};** **// *cung cấp trình khởi tạo mặc định cho thành viên m***



**Y (int a): X {a}, m {a}**

**{Y (): X {0} {};**

**};**

**X g {1}; //*khởi tạo biến toàn cục***

**void f (int a)**

**{**

**X xác định {};**

**Y de2 {};**

**NS∗ p {nullptr};**

**X var {2};**

**p = new X {4};**

**X a [] {1,2,3};**

**vectơ <X> v {1,2,3,4};**

* ***lỗi: không có giá trị mặc định cho X***
* ***OK: sử dụng hàm tạo mặc định***
* ***khởi tạo biến cục bộ*// *khởi tạo đối tượng trên cửa hàng miễn phí*// *khởi tạo các phần tử mảng*// *khởi tạo các phần tử vectơ***

**}**



**492** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Vì lý do này, {} khởi tạo đôi khi được gọi là *phổ quát* khởi tạo: ký hiệu có thể được sử dụng ở mọi nơi. Ngoài ra,{} khởi tạo là *đồng phục*: bất cứ nơi nào bạn khởi tạo một đối tượng kiểu NS với một giá trị v sử dụng {v} ký hiệu, cùng một giá trị của loại NS (X {v}) được tạo ra. Các= và () ký**

**hiệu để khởi tạo (§6.3.5) không phổ biến. Ví dụ:**

**struct Y: X {**

**X m;**

**Y (int a): X (a), m = a {};** **// *lỗi cú pháp: không thể sử dụng = để khởi tạo thành viên***

**};**

**X g (1);** **// *khởi tạo biến toàn cục***

**void f (int a)**

**{**

**X def ();** **// *hàm trả về một X (ngạc nhiên !?)***

**NS∗ p {nullptr};**

**X var = 2;** **// *khởi tạo biến cục bộ*// *lỗi cú pháp: không thể sử***

**p = mới X = 4;** ***dụng = cho mới*// *lỗi: không thể sử dụng () để khởi***

**X a [] (1,2,3);** ***tạo mảng*// *lỗi: không thể sử dụng () cho các phần***

**vectơ <X> v (1,2,3,4);** ***tử danh sách***

**}**

**Các = và () các ký hiệu khởi tạo cũng không đồng nhất, nhưng may mắn thay, các ví dụ về điều đó rất ít người biết đến. Nếu bạn nhấn mạnh vào việc sử dụng= hoặc () khởi tạo, bạn phải nhớ nơi chúng được phép và ý nghĩa của chúng.**

**Các quy tắc giải quyết quá tải thông thường (§12.3) áp dụng cho các hàm tạo. Ví dụ:**

**struct S {**

**S (const char∗); S**

**(gấp đôi∗);**

**};**



**S s1 {"Napier"};**

**S s2 {new double {1.0}};**

**S s3 {nullptr};** **// *không rõ ràng: S :: S (const char \*) hay S :: S (double \*)?***

**Lưu ý rằng {}ký hiệu -initializer không cho phép thu hẹp (§2.2.2). Đó là một lý do khác để thích{} phong cách hơn () hoặc =.**

**17.3.2.1 Khởi tạo bởi Constructors**

**Sử dụng () ký hiệu, bạn có thể yêu cầu sử dụng một hàm tạo trong một lần khởi tạo. Nghĩa là, bạn có thể đảm bảo rằng đối với một lớp, bạn sẽ được khởi tạo bằng phương thức khởi tạo và không nhận được khởi tạo thành viên hoặc khởi tạo danh sách khởi tạo (§17.3.4) mà{} ký hiệu cũng cung cấp. Ví dụ:**

**struct S1 {**

**int a, b;** **// *không có nhà xây dựng***

**};**



|  |  |  |
| --- | --- | --- |
| **Mục 17.3.2.1** | **Khởi tạo bởi Constructors** | **493** |
| **struct S2 {** |  |  |
| **int a, b;** |  |  |
| **S2 (int a = 0, int b = 0): a (aa), b (bb) {}** | **// *constructor*** |  |

**};**

|  |  |  |
| --- | --- | --- |
| **S1 x11** | **(1,2);** | **// *lỗi: không có hàm tạo*// *OK:*** |
| **S1 x12** | **{1,2};** | ***khởi tạo thành viên*** |
| **S1 x13** | **(1);** | **// *error: no constructor*//** |
| **S1 x14** | **{1};** | ***OK: x14.b trở thành 0*** |
| **S2 x21** | **(1,2);** | **// *OK: sử dụng*** |
| **S2 x22** | **{1,2};** | **// *OK: use*** |

**S2 x23 (1);** **// *OK: sử dụng hàm tạo và một đối số mặc định*//**

**S2 x24 {1};** ***OK: use constructor and one default argument***

**Việc sử dụng thống nhất {} chỉ có thể khởi tạo trong C ++ 11, vì vậy mã C ++ cũ hơn sử dụng () và = khởi tạo. Do đó,() and = có thể quen thuộc hơn với bạn. Tuy nhiên, tôi không biết bất kỳ lý do hợp lý nào để thích() ký hiệu ngoại trừ trường hợp hiếm hoi mà bạn cần phân biệt giữa khởi tạo với danh sách các phần tử và danh sách các đối số của phương thức khởi tạo. Ví dụ:**

**vectơ <int> v1 {77};** **// *một phần tử có giá trị 77*// *77 phần***

**vectơ <int> v2 (77);** ***tử có giá trị mặc định 0***

**Vấn đề này - và sự cần thiết phải chọn - có thể xảy ra khi một kiểu có hàm tạo danh sách bộ khởi tạo (§17.3.4), thường là một vùng chứa, cũng có một '' hàm tạo thông thường '' chấp nhận các đối số của kiểu phần tử. Đặc biệt, chúng ta thỉnh thoảng phải sử dụng()khởi tạo cho vectơs của số nguyên và số dấu phẩy động nhưng không bao giờ cần vectơs của chuỗi con trỏ:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **vectơ <chuỗi> v1** | | **{77};** | | **// *77 phần tử có giá trị mặc định ""*** |
| **vectơ <chuỗi> v2** | | **(77);** | |  |
| **vector <string> v3** | | | **{"Booh!"};** | **// *một phần tử có giá trị "Booh!"*// *lỗi: không có*** |
| **vector <string> v4** | | | **("Booh!");** | ***hàm tạo nào nhận đối số chuỗi*** |
| **vectơ <int∗> v5** | **{100,0};** | | | **// *100 int \* s được khởi tạo thành nullptr (100 không phải là int \*)*** |
| **vectơ <int∗> v6** | **{0,0};** | | | **// *2 int \* s được khởi tạo thành*** |
| **vectơ <int∗> v7** | **(0,0);** | | | ***nullptr*// *vectơ rỗng (v7.size () ==*** |
| **vectơ <int∗> v8;** | |  |  | ***0)*// *vectơ rỗng (v7.size () == 0)*** |



**Các v6 và v7 các ví dụ chỉ được các luật sư và người kiểm tra ngôn ngữ quan tâm.**

**17.3.3 Trình tạo mặc định**

**Một hàm tạo có thể được gọi mà không cần đối số được gọi là *nhà xây dựng mặc định*. Các hàm tạo mặc định rất phổ biến. Ví dụ:**



**494** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Vectơ lớp {**

**công cộng:**

**Véc tơ ();** **// *hàm tạo mặc định: không có phần tử***

* ***...***

**};**

**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;**

**Vectơ v2 {};**

* ***VÂNG***
* ***VÂNG***

**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:**

**chuỗi lớp {**

**công cộng:**

**Chuỗi (const char∗ p =** **// *hàm tạo mặc định: chuỗi trống***

**""); //*...***

**};**

**Chuỗi s1;**

**Chuỗi s2 {};**

* ***VÂNG***
* ***VÂNG***

**Thư viện tiêu chuẩn vectơ và dây có các hàm tạo mặc định như vậy (§36.3.2, §31.3.2).**

**Các kiểu dựng sẵn được coi là có các hàm tạo mặc định và sao chép. Tuy nhiên, đối với kiểu dựng sẵn, hàm tạo mặc định không được gọi chotĩnh biến (§17.3). Giá trị mặc định của kiểu tích hợp là0 cho số nguyên, 0,0 cho các loại dấu phẩy động và nullptr cho con trỏ. Ví dụ:**

**void f ()**

**{**

**int a0;** **// *chưa được khởi tạo***



**int a1 ();**

**int a {};**

**gấp đôi d {};** **// *d trở thành 0,0*// *p***

**char∗ P {};** ***trở thành nullptr***

**NS∗ p1 = int mới; NS∗ // *int chưa khởi tạo***

**p2 = new int {};** **// *int được khởi tạo thành 0***

**}**

**Các hàm tạo cho các kiểu dựng sẵn thường được sử dụng nhiều nhất cho các đối số mẫu. Ví dụ:**

**mẫu <lớp T>**

**struct Xử lý {**

**NS∗ P;**

**Xử lý (T∗ pp = new T {}): p {pp} {} //**

***...***

**};**

**Xử lý <int> px;** **// *sẽ tạo int {}***



**Mục 17.3.3** **Trình tạo mặc định** **495**

**Tạo ra NS sẽ được khởi tạo thành 0. Tài liệu tham khảo vàhăng sôs phải được khởi tạo (§7.7, §7.5). Do đó, một lớp chứa các thành viên như vậy không thể được xây dựng mặc định trừ khi người lập trình cung**

**cấp các bộ khởi tạo thành viên trong lớp (§17.4.4) hoặc xác định một hàm tạo mặc định khởi tạo chúng (§17.4.1). Ví dụ:**

**int cầu {9};**

**struct X {**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **const int a1 {7};** | **// *VÂNG*** |  |
|  | **const int a2;** | **// *error: yêu cầu một hàm tạo do người dùng xác định*//** |  |
|  | **const int & r {9};** | ***VÂNG*** |  |
|  | **int & r1 {global};** |  |  |
| **};** | **int & r2;** |  |  |
|  |  |  |
| **X x;** | **// *lỗi: không có hàm tạo mặc định cho X*** | |  |

**Một mảng, một thư viện chuẩn vectơvà các vùng chứa tương tự có thể được khai báo để cấp phát một số phần tử được khởi tạo mặc định. Trong những trường hợp như vậy, một phương thức khởi tạo mặc định rõ ràng là bắt buộc đối với một lớp được sử dụng làm loại phần tử củavectơ hoặc mảng. Ví dụ:**

|  |  |  |  |
| --- | --- | --- | --- |
| **struct S1 {S1 (); }; struct** | | | **// *có hàm tạo mặc định*//** |
| **S2 {S2 (string); };** | | | ***không có hàm tạo mặc định*** |
| **S1 a1** | **[10];** |  | **// *OK: 10 phần tử mặc định*// *lỗi: không thể*** |
| **S2 a2** | **[10];** |  | ***khởi tạo các phần tử*// *OK: hai phần tử: S2*** |
| **S2 a3** | **[] {"alpha", "beta"};** | | ***{"alpha"}, S2 {"beta"}*** |
| **vectơ <S1> v1 (10);** | | | **// *OK: 10 phần tử mặc định*// *lỗi: không thể*** |
| **vectơ <S2> v2** | | **(10);** | ***khởi tạo các phần tử*// *OK: hai phần tử: S2*** |
| **vectơ <S2> v3** | | **{"alpha", "beta"};** | ***{"alpha"}, S2 {"beta"}*** |



**vectơ <S2> v2 (10, "");**

**vectơ <S2> v4;**

**Khi nào một lớp nên có một phương thức khởi tạo mặc định? Một câu trả lời kỹ thuật đơn giản là '' khi bạn sử dụng nó làm loại phần tử cho một mảng, v.v. '' Tuy nhiên, câu hỏi hay hơn là '' Đối với những loại nào thì việc có giá trị mặc định là hợp lý? '' Hoặc thậm chí '' Loại này có giá trị 'đặc biệt' mà chúng ta có thể 'tự nhiên' sử dụng làm mặc định không? '' Chuỗi có chuỗi trống,"", vùng chứa có tập hợp trống, {}và các giá trị số có số không. Rắc rối khi quyết định mặc địnhNgày (§16.3) phát sinh vì không có ngày mặc định '' tự nhiên '' (vụ nổ Big Bang đã quá xa trong quá khứ và không được liên kết chính xác với ngày hàng ngày của chúng ta). Bạn không nên quá khéo léo khi phát minh ra các giá trị mặc định. Ví dụ: vấn đề với vùng chứa các phần tử không có giá trị mặc định thường được giải quyết tốt nhất bằng cách không phân bổ các phần tử cho đến khi bạn có các giá trị thích hợp cho chúng (ví dụ: sử dụngpush\_back ()).**

**17.3.4 Trình tạo danh sách khởi tạo**

**Một hàm tạo nhận một đối số duy nhất của kiểu std :: initializer\_list được gọi là *phương thức khởi tạo danh sách khởi tạo*. Một phương thức khởi tạo danh sách khởi tạo được sử dụng để xây dựng các đối tượng bằng cách sử dụng{}-danh sách làm trình khởi tạo của nó**



**496** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**giá trị. Vùng chứa thư viện tiêu chuẩn (ví dụ:vectơ và bản đồ) có các hàm tạo danh sách khởi tạo, các phép gán, v.v. (§31.3.2, §31.4.3). Xem xét:**

**vector <double> v = {1, 2, 3,456, 99,99};**

**danh sách <cặp <chuỗi, chuỗi >> ngôn ngữ = {**

**{"Nygaard", "Simula"}, {"Richards", "BCPL"}, {"Ritchie", "C"}**

**};**

**bản đồ <vector <string>, vector <int>> years = {**

**{{"Maurice", "Vincent", "Wilkes"}, {1913, 1945, 1951, 1967, 2000}},**

**{{"Martin", "Richard**

**{{"David", "John", "**

**};**

**Cơ chế chấp nhận một {}-list là một hàm (thường là một hàm tạo) nhận đối số kiểu**

**std :: initializer\_list <T>. Ví dụ:void f**

**(khởi tạo\_danh sách <int>);**

* **({1,2});**
* **({23,345,4567,56789});**

**NS({});// *danh sách trống***

**f {1,2};** **// *error: function call () bị thiếu***

**years.insert ({{"Bjarne", "Stroustrup"}, {1950, 1975, 1985}});**

**Danh sách bộ khởi tạo có thể có độ dài tùy ý nhưng phải đồng nhất. Nghĩa là, tất cả các phần tử phải thuộc loại đối số mẫu,NShoặc hoàn toàn có thể chuyển đổi thành NS.**

**17.3.4.1 initializer\_list Định hướng cấu trúc**



**Khi bạn có vài trò lừa đảo**

**để chọn đối số phù hợp cho một tập hợp các đối số nhất định. Để chọn một phương thức khởi tạo, danh sách mặc định và trình khởi tạo sẽ được ưu tiên. Xem xét:**

**struct X {**

**X (khởi tạo\_danh sách <gợi ý>);**

**NS();**

**X (int);**

**};**

**X x0 {};** **// *danh sách trống: phương thức khởi tạo mặc định hay phương thức khởi tạo danh sách? (hàm tạo mặc***

**X x1 {1};** ***định)*// *một số nguyên: một đối số int hay danh sách một phần tử? (hàm khởi tạo danh sách khởi tạo)***

**Các quy tắc là:**

* **Nếu một hàm tạo mặc định hoặc một hàm tạo danh sách khởi tạo có thể được gọi, hãy ưu tiên hàm tạo mặc định.**
* **Nếu có thể gọi cả hàm khởi tạo danh sách bộ khởi tạo và '' hàm tạo thông thường '', hãy ưu tiên hàm tạo danh sách bộ khởi tạo.**



**Mục 17.3.4.1** **initializer\_list Định hướng cấu trúc** **497**

**Quy tắc đầu tiên, '' ưu tiên hàm tạo mặc định '', về cơ bản là cách hiểu thông thường: hãy chọn hàm tạo đơn giản nhất khi bạn có thể. Hơn nữa, nếu bạn xác định một phương thức khởi tạo danh sách khởi tạo để thực hiện điều gì đó với một danh sách trống khác với những gì mà phương thức khởi tạo mặc định thực hiện, có thể bạn đã mắc phải lỗi thiết kế.**

**Quy tắc thứ hai, '' ưu tiên phương thức khởi tạo danh sách trình khởi tạo '', là cần thiết để tránh các độ phân giải khác nhau dựa trên số lượng phần tử khác nhau. Xem xétstd :: vector (§31.4):**

|  |  |  |
| --- | --- | --- |
| **vectơ <int> v1** | **{1};** | **// *một phần tử*** |
| **vectơ <int> v2** | **{1,2};** | **// *hai yếu tố*** |
| **vectơ <int> v3** | **{1,2,3};** | **// *ba yếu tố*** |

**vector <string> vs1 {"one" vector <string> vs2**

**{"one" vector <string> vs3 {"one", "two",**

**"three"};**

**Trong mọi trường hợp, phương thức khởi tạo danh sách khởi tạo được sử dụng. Nếu chúng ta thực sự muốn gọi hàm tạo nhận một hoặc hai đối số số nguyên, chúng ta phải sử dụng() ký hiệu:**

**vectơ <int> v1 (1); //*một phần tử có giá trị mặc định (0)*vectơ**

**<int> v2 (1,2); //*một phần tử có giá trị 2***

**17.3.4.2 Sử dụng initializer\_listNS**

**Một chức năng với một initializer\_list <T> đối số có thể truy cập nó dưới dạng một chuỗi bằng cách sử dụng các hàm thành**

**viên bắt đầu(), kết thúc(), và kích thước(). Ví dụ:**

**void f (initializer\_list <int> args) {**

**for (int i = 0; i! = args.size (); ++ i)**

**cout << args.begin () [i] << "\ n";**

**}**



**Không may, initializer\_list**

**Một initializer\_list <T> Là**

**(§12.3) và không áp đặt chi phí vì initializer\_list <T> đối tượng chỉ là một xử lý nhỏ (thường là hai từ) cho một mảng NSNS.**

**Vòng lặp đó có thể tương đương đã được viết:**

**void f (initializer\_list <int> args) {**

**for (auto p = args.begin (); p! = args.end (); ++ p)**

**cout << ∗p << "\ n";**

**}**

**hoặc:**

**void f (initializer\_list <int> args) {**

**for (auto x: args)**

**cout << x << "\ n";**

**}**



**498** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Để sử dụng một cách rõ ràng initializer\_list bạn phải #bao gồm tệp tiêu đề trong đó nó được xác định: <ban đầu-izer\_list>. Tuy nhiên, kể từvectơ, bản đồ, v.v., sử dụng initializer\_lists, tiêu đề của chúng (<giám đốc>, <bản đồ>, Vân vân.)**

**đã sẵn sàng #bao gồm <initializer\_list>, vì vậy bạn hiếm khi phải làm như vậy trực tiếp. Các yếu tố của mộtinitializer\_list là bất biến. Thậm chí đừng nghĩ đến việc cố gắng sửa đổi các giá trị của họ. Ví dụ:**

**int f (std :: initializer\_list <int> x, int val) {**

**∗x.begin () = val;** **// *error: cố gắng thay đổi giá trị của phần tử danh sách trình khởi tạo***

**trở lại∗x.begin ();** **// *VÂNG***

**}**

**void g ()**

**{**

**for (int i = 0; i! = 10; ++ i)**

**cout << f ({1,2,3}, i) << '\ n';**

**}**

**Có nhiệm vụ trong NS() đã thành công, có vẻ như giá trị của 1 (trong {1,2,3}) có thể thay đổi. Điều đó sẽ gây ra thiệt hại nghiêm trọng cho một số khái niệm cơ bản nhất của chúng ta. Tại vìinitializer\_list các phần tử là bất biến, chúng ta không thể áp dụng một hàm tạo chuyển động (§3.3.2, §17.5.2) cho chúng.**

**Một vùng chứa có thể triển khai một phương thức khởi tạo danh sách khởi tạo như thế này:**

**mẫu <lớp E>**

**Vectơ lớp {**

**công cộng:**

**Vector (std :: initializer\_list <E> s); //*phương thức khởi tạo danh sách khởi tạo*// *...***

**riêng:**

**int sz;**



**E∗ độ cao;**

**};**

**mẫu <lớp E>**

**Vector :: Vector (std :: initializer\_list <E> s)**

**: sz {s.size ()}**

**{**

* ***đặt kích thước vector***

**dự trữ (sz);** **// *có được lượng không gian phù hợp***

**uninitialized\_copy (s.begin (), s.end (), elem);** **// *khởi tạo các phần tử trong elem [0: s.size ())***

**}**

**Danh sách trình khởi tạo là một phần của thiết kế khởi tạo phổ biến và thống nhất (§17.3).**

**17.3.4.3 Khởi tạo trực tiếp và sao chép**

**Sự khác biệt giữa khởi tạo trực tiếp và khởi tạo sao chép (§16.2.6) được duy trì cho {}khởi tạo. Đối với một vùng chứa, điều này ngụ ý rằng sự phân biệt được áp dụng cho cả vùng chứa và các phần tử của nó:**



**Mục 17.3.4.3** **Khởi tạo trực tiếp và sao chép** **499**

* **Hàm khởi tạo danh sách bộ khởi tạo của vùng chứa có thể là rõ ràng hay không.**
* **Hàm tạo của kiểu phần tử của danh sách trình khởi tạo có thể là rõ ràng hay không.**

**Cho một vector <vector <double>>, chúng ta có thể thấy sự khác biệt giữa khởi tạo trực tiếp so với khởi tạo sao chép được áp dụng cho các phần tử. Ví dụ:**

**vectơ <vectơ <double>> vs = {**

**{10,11,12,13,14},** **// *OK: vectơ của năm phần tử*// *OK:***

**{10},** ***vectơ của một phần tử*// *error: vector***

**10,** ***<double> (int) là rõ ràng***

**vector <double> {10,11,12,13},** **// *OK: vectơ của năm phần tử***

**vector <double> {10},**

**vectơ <double> (10),**

**};**

**Một vùng chứa có thể có một số hàm tạo rõ ràng và một số thì không.** **Thư viện tiêu chuẩn vectơ là một**

**ví dụ về điều đó. Ví dụ,std :: vector <int> (int) Là rõ ràng, nhưng std :: vector <int> (initialize\_list <int>)**

**không phải:**

**vectơ <đôi> v1 (7);** **// *OK: v1 có 7 phần tử; lưu ý: use () thay vì {}*// *lỗi:***

**vectơ <đôi> v2 = 9;** ***không có chuyển đổi từ int sang vector***

**void f (const vector <đôi> &);**

**void g ()**

**{**

**v1 = 9;** **// *lỗi: không có chuyển đổi từ int sang vector***

**f (9);** **// *lỗi: không có chuyển đổi từ int sang vector***

**}**

**Bằng cách thay thế () với {} chúng tôi nhận được:**

**vectơ <đôi> v1 {7};** **// *OK: v1 có một phần tử (với giá trị 7)***



**vectơ <đôi> v2 = {9};**

**void f (const vector <nghi**

**void g ()**

**{**

**v1 = {9};** **// *OK: v1 bây giờ có một phần tử (với giá trị 9)*// *OK: f***

**f ({9});** ***được gọi với danh sách {9}***

**}**

**Rõ ràng, các kết quả là khác nhau đáng kể.**

**Ví dụ này được xây dựng cẩn thận để đưa ra một ví dụ về các trường hợp khó hiểu nhất. Lưu ý rằng sự mơ hồ rõ ràng (trong mắt người đọc chứ không phải trình biên dịch) không xuất hiện trong danh sách dài hơn. Ví dụ:**

**vectơ <đôi> v1 {7,8,9}; vectơ** **// *OK: v1 có ba phần tử với các giá trị {7,8,9}*//**

**<đôi> v2 = {9,8,7};** ***OK: v2 có ba phần tử với các giá trị {9,8,7}***



**500** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**void f (const vector <đôi> &);**

**void g ()**

**{**

**v1 = {9,10,11};** **// *OK: v1 bây giờ có ba phần tử với các giá trị {9,10,11}*//**

**f ({9,8,7,6,5,4});** ***OK: f được gọi với danh sách {9,8,7,6,5,4}***

**}**

**Tương tự, sự không rõ ràng tiềm ẩn không xảy ra đối với danh sách các phần tử thuộc loại không chung:**

**vector <string> v1 {"Anya"}; vector** **// *OK: v1 có một phần tử (với giá trị "Anya")*// *OK: v2***

**<string> v2 = {"Courtney"};** ***có một phần tử (với giá trị "Courtney")***

**void f (const vector <string**

**void g ()**

**{**

**v1 = {"Gavin"};** **// *OK: v1 bây giờ có một phần tử (với giá trị "Gavin")*// *OK: f***

**f ({"Norah"});** ***được gọi với danh sách {"Norah"}***

**}**

**17.4 Khởi tạo thành viên và cơ sở**

**Các nhà xây dựng có thể thiết lập các bất biến và thu được các tài nguyên. Nói chung, chúng làm điều đó bằng cách khởi tạo các thành viên lớp và các lớp cơ sở.**

**17.4.1 Khởi tạo thành viên**

**Hãy xem xét một lớp học có thể được sử dụng để giữ thông tin cho một tổ chức nhỏ:**

**Câu lạc bộ lớp học {**

**tên chuỗi;**



**các thành viên vector <string>;**

**Văn phòng vector <string>**

**Ngày thành lập;**

* ***...***

**Câu lạc bộ (const string & n, Date fd);**

**};**

**Các Câu lạc bộHàm tạo của câu lạc bộ lấy tên của câu lạc bộ và ngày thành lập của nó làm đối số. Các đối số cho hàm tạo của một thành viên được chỉ định trong một*danh sách khởi tạo thành viên* trong định nghĩa về hàm tạo của lớp chứa. Ví dụ:**

**Câu lạc bộ :: Câu lạc bộ (const string & n, Date fd)**

**: name {n}, thành viên {}, sĩ quan {}, thành lập {fd}**

**{**

* ***...***

**}**

**Danh sách trình khởi tạo thành viên bắt đầu bằng dấu hai chấm và các trình khởi tạo thành viên riêng lẻ được phân tách bằng dấu phẩy.**



**Mục 17.4.1** **Khởi tạo thành viên** **501**

**Các hàm tạo của các thành viên được gọi trước khi phần thân của hàm tạo riêng của lớp chứa được thực thi (§17.2.3). Các hàm tạo được gọi theo thứ tự mà các thành viên được khai báo trong lớp chứ không phải thứ tự mà các thành viên xuất hiện trong danh sách trình khởi tạo. Để tránh nhầm lẫn, cách tốt nhất là chỉ định các bộ khởi tạo trong thứ tự khai báo thành viên. Hy vọng về một cảnh báo của trình biên dịch nếu bạn không làm đúng thứ tự. Các hàm hủy thành viên được gọi theo thứ tự xây dựng ngược lại sau khi phần thân của hàm hủy của chính lớp đó đã được thực thi.**

**Nếu một phương thức khởi tạo thành viên không cần đối số, thì thành viên đó không cần được đề cập đến trong danh sách khởi tạo thành viên. Ví dụ:**

**Câu lạc bộ :: Câu lạc bộ (const string & n, Date fd)**

**: name {n}, thành lập {**

**{**

* ***...***

**}**

**Hàm tạo này tương đương với phiên bản trước. Trong mỗi trường hợp,Câu lạc bộ :: sĩ quan và Câu lạc bộ :: thành viên được khởi tạo thành một vectơ không có phần tử nào.**

**Thông thường nên rõ ràng về việc khởi tạo các thành viên. Lưu ý rằng thành viên '' được khởi tạo ngầm định '' của kiểu tích hợp không được khởi tạo (§17.3.1).**

**Một phương thức khởi tạo có thể khởi tạo các thành viên và cơ sở của lớp nó, nhưng không phải là thành viên hoặc cơ sở của các thành viên hoặc cơ sở của nó. Ví dụ:**

**struct B {B (int); / \**...* \* /};**

**struct BB: B {/ \**...* \* /};**

**struct BBB: BB {**

1. **(int i): B (i) {}; //*lỗi: cố gắng khởi tạo cơ sở của base*//**

***...***

**};**

**17.4.1.1 Khởi tạo và chỉ định thành viên**



**Các bộ khởi tạo thành viên là**

**bài tập của nhiệm vụ. Ví dụ:**

**lớp X {**

**const int i;**

**Câu lạc bộ cl;**

**Câu lạc bộ & rc;**

* ***...***

**X (int ii, const string & n, Date d, Club & c): i {ii}, cl {n, d}, rc {c} {}**

**};**

**Một thành viên tham khảo hoặc một hăng sô thành viên phải được khởi tạo (§7.5, §7.7, §17.3.3). Tuy nhiên, đối với hầu hết các kiểu lập trình viên có sự lựa chọn giữa việc sử dụng bộ khởi tạo và sử dụng một phép gán. Trong trường hợp đó, tôi thường thích sử dụng cú pháp của trình khởi tạo thành viên để làm rõ rằng quá trình khởi tạo đang được thực hiện. Thông thường, cũng có một lợi thế hiệu quả khi sử dụng cú pháp trình khởi tạo (so với việc sử dụng một phép gán). Ví dụ:**



**502** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Người trong lớp {**

**tên chuỗi;**

**địa chỉ chuỗi;**

* ***...***

**Người (const Person &);**

**Person (const string & n, const string & a);**

**};**

**Person :: Person (const string & n, const string & a)**

**: tên {n}**

**{**

**địa chỉ = a;**

**}**

* **đây Tên được khởi tạo bằng một bản sao của n. Mặt khác,Địa chỉ đầu tiên được khởi tạo thành chuỗi trống và sau đó là bản sao của Một được gán.**

**17.4.2 Bộ khởi tạo cơ sở**

**Các cơ sở của một lớp dẫn xuất được khởi tạo giống như cách các thành viên không phải là dữ liệu. Nghĩa là, nếu một cơ sở yêu cầu một bộ khởi tạo, nó phải được cung cấp như một bộ khởi tạo cơ sở trong một phương thức khởi tạo. Nếu chúng ta muốn, chúng ta có thể chỉ định rõ ràng cấu trúc mặc định. Ví dụ:**

**lớp B1 {B1 (); }; hạng B2//{B2*có hàm*(int); *tạo*}//*khôngmặcđịnhcó***

***hàm tạo mặc định***

**struct D1: B1, B2 {**

**D1 (int i): B1 {}, B2 {i} {}**

**};**

**struct D2: B1, B2 {**



**D2 (int i): B2 {i} {}**

**};**

**struct D1: B1, B2 {**

**D1 (int i) {}** **// *lỗi: B2 yêu cầu bộ khởi tạo int***

**};**

**Như với các thành viên, thứ tự khởi tạo là thứ tự khai báo và nên chỉ định các trình khởi tạo cơ sở theo thứ tự đó. Các căn cứ được khởi tạo trước khi có thành viên và bị hủy sau khi có thành viên (§17.2.3).**

**17.4.3 Ủy quyền xây dựng**

**Nếu bạn muốn hai hàm tạo thực hiện cùng một hành động, bạn có thể tự lặp lại hoặc xác định '' an trong đó()chức năng '' để thực hiện hành động chung. Cả hai '' giải pháp '' đều phổ biến (vì các phiên bản C ++ cũ hơn không cung cấp bất kỳ điều gì tốt hơn). Ví dụ:**



**Mục 17.4.3** **Ủy quyền người xây dựng** **503**

**lớp X {**

**int a;**

**validate (int x) {if (0 <x && x <= max) a = x; khác ném Bad\_X (x); } công cộng:**

**X (int x) {validate (x); } X**

**() {validate (42); }**

**X (string s) {int x = to <int> (s); xác thực (x); } // // *§25.2.5.1***

***...***

**};**

**Độ dài dòng cản trở khả năng đọc và việc lặp lại dễ xảy ra lỗi. Cả hai đều cản trở khả năng bảo trì. Cách thay thế là xác định o**

**lớp X {**

**int a;**

**công cộng:**

**X (int x) {if (0 <x && x <= max) a = x; khác ném Bad\_X (x); } X ():**

**X {42} {}**

|  |  |
| --- | --- |
| **X (string s): X {to <int> (s)}** | **// *§25.2.5.1*** |
| **{} // *...*** |  |

**};**

**Nghĩa là, một trình khởi tạo kiểu thành viên sử dụng tên riêng của lớp (tên phương thức khởi tạo của nó) gọi một phương thức khởi tạo khác như một phần của cấu trúc. Một hàm tạo như vậy được gọi là*ủy quyền người xây dựng* (và thỉnh thoảng một *nhà xây dựng chuyển tiếp*).**

**Bạn không thể vừa ủy quyền vừa khởi tạo rõ ràng một thành viên. Ví dụ:**

**lớp X {**

**int a;**

**công cộng:**

**X (int x) {if (0 <x && x <= max) a = x; khác ném Bad\_X (x); }**



**X (): X {42}, a {56} {} //*lỗi***

* ***...***

**};**

**Việc ủy quyền bằng cách gọi một phương thức khởi tạo khác trong danh sách thành viên và bộ khởi tạo cơ sở của một phương thức khởi tạo rất khác với việc gọi một cách rõ ràng một phương thức khởi tạo trong phần thân của một phương thức khởi tạo. Xem xét:**

**lớp X {**

**int a;**

**công cộng:**

**X (int x) {if (0 <x && x <= max) a = x; khác ném Bad\_X (x); } X () {X**

**{42}; } //*có khả năng xảy ra lỗi***

* ***...***

**};**

**Các X {42} chỉ cần tạo một đối tượng không tên mới (tạm thời) và không làm gì với nó. Việc sử dụng như vậy thường xuyên hơn không phải là một lỗi. Hy vọng về một cảnh báo trình biên dịch.**

**Một đối tượng không được coi là đã xây dựng cho đến khi phương thức khởi tạo của nó hoàn thành (§6.4.2). Khi sử dụng một phương thức khởi tạo ủy quyền, đối tượng không được coi là đã xây dựng cho đến khi phương thức khởi tạo ủy quyền hoàn thành - việc chỉ hoàn thành hàm tạo ủy quyền là không đủ. Một trình hủy sẽ không**



**504** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**được gọi cho một đối tượng trừ khi phương thức khởi tạo ban đầu của nó đã hoàn thành.**

**Nếu tất cả những gì bạn cần là đặt một thành viên thành giá trị mặc định (không phụ thuộc vào đối số của hàm tạo), thì bộ khởi tạo thành viên (§17.4.4) có thể đơn giản hơn.**

**17.4.4 Bộ khởi tạo trong lớp**

**Chúng tôi có thể chỉ định một trình khởi tạo cho mộttĩnh thành viên dữ liệu trong khai báo lớp. Ví dụ:**

**lớp A {**

**công cộng:**

**int a {7};**

**int b = 77;**

**};**

**Vì những lý do kỹ thuật khá mù mờ liên quan đến phân tích cú pháp và tra cứu tên, {} và = ký hiệu bộ khởi tạo có thể được sử dụng cho bộ khởi tạo thành viên trong lớp, nhưng () ký hiệu không thể.**

**Theo mặc định, một phương thức khởi tạo sẽ sử dụng một trình khởi tạo trong lớp như vậy, vì vậy ví dụ đó tương đương với:**

**lớp A {**

**công cộng:**

**int a;**

**int b;**

**A (): a {7}, b {77} {}**

**};**

**Việc sử dụng các bộ khởi tạo trong lớp như vậy có thể tiết kiệm một chút công việc nhập, nhưng lợi ích thực sự đến với các lớp phức tạp hơn với nhiều hàm tạo. Thông thường, một số hàm tạo sử dụng cùng một bộ khởi tạo cho một thành viên. Ví dụ:**

**lớp A {**

**công cộng:**

**A (): a {7}, b {5}, thuật toán {"MD5"}, trạng thái {"Constructor run"} {} A**



**(int a\_val): a {a\_val}**

**A (D d): a {7}, b {g (d)}, //**

***...***

**riêng:**

**int a, b;**

**Thuật toán HashFunction; trạng** **// *băm mật mã được áp dụng cho tất cả As*// *chuỗi***

**thái chuỗi;** ***biểu thị trạng thái trong vòng đời đối tượng***

**};**

**Thực tế là thuật toán và tiểu bang có cùng giá trị trong tất cả các hàm tạo bị mất trong mớ mã và có thể dễ dàng trở thành vấn đề bảo trì. Để làm cho các giá trị chung trở nên rõ ràng, chúng ta có thể đưa ra trình khởi tạo duy nhất cho các thành viên dữ liệu:**

**lớp A {**

**công cộng:**

**A (): a {7}, b {5} {}**

**A (int a\_val): a {a\_val}, b {5} {} A**

**(D d): a {7}, b {g (d)} {}**

**// *...***



|  |  |  |
| --- | --- | --- |
| **Mục 17.4.4** | **Bộ khởi tạo trong lớp** | **505** |
| **riêng:** |  |  |
| **int a, b;** |  |  |
| **Thuật toán HashFunction {"MD5"};** | **// *băm mật mã được áp dụng cho tất cả As*// *chuỗi*** |  |
| **trạng thái chuỗi {"Khối mã lệnh chạy"};** | ***biểu thị trạng thái trong vòng đời đối tượng*** |  |

**};**

**Nếu một thành viên được khởi tạo bởi cả trình khởi tạo trong lớp và một phương thức khởi tạo, thì chỉ việc khởi tạo của phương thức khởi tạo được thực hiện (nó '' ghi đè '' mặc định). Vì vậy, chúng tôi có thể đơn giản hóa thêm:**

**lớp A {**

**công cộng:**

**MỘT() {}**

**A (int a\_val): a {a\_val}**

**A (D d): b {g (d)} {} //**

***...***

**riêng:**

**int a {7};**

**int b {5};**

**Thuật toán HashFunction {"MD5"};**

**trạng thái chuỗi {"Khối mã lệnh chạy"};**

**// *ý nghĩa của 7 đối với a là ...*// *ý nghĩa của 5 đối* *với b là ...*// *Băm mật mã được áp dụng cho tất cả As*// *Chuỗi biểu thị trạng thái trong vòng đời đối tượng***

**};**

**Như đã trình bày, các trình khởi tạo trong lớp mặc định cung cấp cơ hội cho tài liệu về các trường hợp phổ biến.**

**Trình khởi tạo thành viên trong lớp có thể sử dụng các tên có phạm vi tại điểm sử dụng chúng trong khai báo thành viên. Hãy xem xét ví dụ kỹ thuật gây nhức đầu sau:**

**int count = 0;**

**int count2 = 0;**

**int f (int i) {return i + count; }**

**struct S {**



**int m1 {count2};**

**int m2 {f (m1)};**

**S () {++ count2; }**

**};**

**int main ()**

**{**

**Đ s1;**

**// *{0,0}***

**+ + đếm;**

**Đ s2;**

**// *{1,2}***

**}**

**Khởi tạo thành viên được thực hiện theo thứ tự khai báo (§17.2.3), vì vậy trước tiên m1 được khởi tạo thành giá trị của một biến toàn cục count2. Giá trị của biến toàn cục nhận được tại điểm mà hàm tạo cho một biến mớiNS đối tượng được chạy, vì vậy nó có thể (và trong ví dụ này là) thay đổi. Kế tiếp,m2 được khởi tạo bởi một cuộc gọi đến toàn cầu NS().**

**Đó là một ý tưởng tồi nếu ẩn các phụ thuộc tinh vi vào dữ liệu toàn cục trong các trình khởi tạo thành viên.**



**506** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**17.4.5 tĩnh Khởi tạo thành viên**

**MỘT tĩnh thành viên lớp được cấp phát tĩnh chứ không phải là một phần của mỗi đối tượng của lớp. Nói chung,tĩnh khai báo thành viên hoạt động như một khai báo cho một định nghĩa bên ngoài lớp. Ví dụ:**

**Nút lớp {**

**// *...***

**static int node\_count;** **// *tuyên ngôn***

**};**

**int Node :: node\_count = 0;** **// *Định nghĩa***

**Tuy nhiên, đối với một số tách**

**đơn giản. Cáctĩnh membe**

**kiểu chữ (§10.4.3) và trình khởi tạo phải là *biểu thức hằng số*. Ví dụ:**

**lớp học Tò mò {**

**công cộng:**

**static const int c1 = 7;** **// *VÂNG***

**static int c2 = 11;** **// *lỗi: không phải const*// *OK, nhưng không tĩnh (§17.4.4)*// *lỗi: trình khởi tạo***

**const int c3 = 13;** ***trong lớp không phải là hằng số*// *lỗi: trong lớp không tích phân (sử dụng***

**static const int c4 = sqrt (9);** ***constexpr thay vì const)***

**static const float c5 = 7,0; //*...***

**};**

**Nếu (và chỉ khi) bạn sử dụng một thành viên đã khởi tạo theo cách yêu cầu nó được lưu trữ như một đối tượng trong bộ nhớ, thì thành viên đó phải được định nghĩa (duy nhất) ở một nơi nào đó. Trình khởi tạo có thể không được lặp lại:**

**const int Curious :: c1;** **// *đừng lặp lại trình khởi tạo ở đây*// *OK:***

**const int∗ p = & Tò mò :: c1;** ***Curious :: c1 đã được xác định***

**Công dụng chính của các hằng thành viên là cung cấp các tên tượng trưng cho các hằng cần thiết ở những nơi khác trong khai báo lớp. Cho người yêu cũ**



**template <class T, int N> class Cố định**

**{// *fimảng kích thước xed*công cộng:**

**static constexpr int max = N; //**

***...***

**riêng:**

**T a [tối đa];**

**};**

**Đối với số nguyên, điều tra viên (§8.4) cung cấp một giải pháp thay thế để xác định các hằng biểu tượng trong một khai báo lớp. Ví dụ:**

**lớp X {**

**enum {c1 = 7, c2 = 11, c3 = 13, c4 = 17}; //*...***

**};**



**Mục 17.5** **Sao chép và di chuyển** **507**

**17.5 Sao chép và Di chuyển**

**Khi chúng ta cần chuyển một giá trị từ Một đến NS, chúng tôi thường có hai tùy chọn riêng biệt về mặt logic:**

* ***Sao chép* là nghĩa thông thường củax = y; nghĩa là, hiệu quả là các giá trị củaNSvàycả hai đều bằngygiá trị của trước khi chuyển nhượng.**
* ***Di chuyển* láNSvớiygiá trị cũ của vàyvới một số *chuyển từ trạng thái.* Đối với các trường hợp thú vị nhất,vùng chứa, trạng thái được chuyển đến 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 là của**

**nên để nguồn o**

**bị phá hủy và bộ hủy không thể hủy một đối tượng còn lại trong trạng thái không hợp lệ. Ngoài ra, các thuật toán thư viện tiêu chuẩn dựa vào việc có thể gán cho (sử dụng di chuyển hoặc sao chép) một đối tượng được chuyển đến. Vì vậy, hãy thiết kế các bước di chuyển của bạn để không ném và để các đối tượng nguồn của chúng ở trạng thái cho phép phá hủy và chuyển nhượng.**

**Để cứu 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 học NS được xác định bởi hai phép toán:**

* **Copy constructor: X (const X &)**
* **Sao chép bài tập: X & operator = (const X &)**

**Bạn có thể xác định hai thao tác này bằng các loại đối số mạo hiểm hơn, chẳng hạn như X dễ bay hơi &, nhưng đừng; bạn sẽ chỉ nhầm lẫn chính bạn và những người khác. Một hàm 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 dụngconst X & là kiểu trả về của phép gán bản sao. Ý kiến của tôi là làm như vậy gây ra nhiều nhầm lẫn hơn mức đáng có, vì vậy cuộc thảo luận của tôi về bản sao giả định rằng hai phép toán có các kiểu thông thường.**



**Hãy xem xét một hai-d đơn giản**

**mẫu <lớp T>**

**ma trận lớp {**

|  |  |  |  |
| --- | --- | --- | --- |
| **mảng <int, 2> dim;** | **// *hai kích thước*** |  |  |
| **NS∗ độ cao;** | **// *con trỏ để làm mờ [0] \* dim [1] các phần tử thuộc loại T*** | |  |
| **công cộng:** |  |  |  |
| **Ma trận (int d1, int d2): dim {d1, d2}, elem {new T [d1∗d2]} {}** | | | **// *đơn giản hóa (không xử lý lỗi)*** |
| **int size () const {return dim [0]∗mờ [1]; }** | | |  |
| **Ma trận (const Matrix &);** | | **// *sao chép hàm tạo*** |  |
| **Ma trận & operator = (const Matrix &);** | | **// *sao chép bài tập*** |  |
| **Ma trận (Matrix &&);** |  | **// *di chuyển hàm tạo*** |  |
| **Ma trận & operator = (Ma trận &&);** | | **// *chuyển nhiệm vụ*** |  |

**˜Matrix () {xóa [] elem; } //**

***...***

**};**



**508** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Đầu tiên, chúng tôi lưu ý rằng bản sao mặc định (sao chép các thành viên) sẽ bị sai nghiêm trọng: Ma trận các phần tử sẽ không được sao chép, Ma trận bản sao sẽ có một con trỏ đến các phần tử giống như nguồn và Ma trận hàm hủy sẽ xóa các phần tử (được chia sẻ) hai lần (§3.3.1).**

**Tuy nhiên, lập trình viên có thể xác định bất kỳ ý nghĩa phù hợp nào cho các hoạt động sao chép này và quy tắc thông thường cho vùng chứa là sao chép các phần tử được chứa:**

**mẫu <lớp T>**

**Ma trận :: Ma trận (const Matrix & m)** **// *sao chép hàm tạo***

* **dim {m.dim},**

**elem {new T [m.size ()]}**

**{**

**uninitialized\_copy (m**

**}**

**mẫu <lớp T>**

**Ma trận & Ma trận :: operator = (const Matrix & m) {** **// *sao chép bài tập***

**if (dim [0]! = m.dim [0] || dim [1]! = m.dim [1])**

**ném runtime\_error ("kích thước xấu trong Matrix ="); sao chép**

**(m.elem, m.elem + m.size (), elem); //*sao chép các yếu tố***

**}**

**Một hàm tạo sao chép và một phép gán bản sao khác nhau ở chỗ một hàm tạo sao chép khởi tạo bộ nhớ chưa được 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.**

**Các Ma trận toán tử gán bản sao có thuộc tính rằng nếu một bản sao của một phần tử ném ra một ngoại lệ, thì mục tiêu của phép gán có thể bị bỏ lại với một hỗn hợp giá trị cũ và giá trị mới. Vậy thôi đoMa trận nhiệm vụ cung cấp sự đảm bảo cơ bản, nhưng không phải là sự đảm bảo mạnh mẽ (§13.2). Nếu điều đó không được coi là chấp nhận được, chúng ta có thể tránh điều đó bằng kỹ thuật cơ bản là tạo bản sao trước rồi hoán đổi các hình đại diện:**



**Ma trận & Ma trận :: operator = {**

**Ma trận tmp {m};**

**// *tạo một bản sao***

**hoán đổi (tmp,∗cái này);**

**// *hoán đổi đại diện của tmp với \* cái này***

**trở lại ∗cái này;**

**}**

**Các tráo đổi() sẽ chỉ được thực hiện nếu sao chép thành công. Rõ ràng, điều nàytoán tử = () chỉ hoạt động nếu việc triển khai tráo đổi() không sử dụng chỉ định (std :: swap () không làm); xem §17.5.2.**

**Thông thường, một phương thức khởi tạo sao chép phải sao chép mọitĩnh thành viên (§17.4.1). Nếu một phương thức khởi tạo sao chép không thể sao chép một phần tử (ví dụ: vì nó cần lấy một tài nguyên không có sẵn để làm như vậy), nó có thể đưa ra một ngoại lệ.**

**Lưu ý rằng tôi đã không bảo vệ Ma trậnsự phân công sao chép chống lại sự tự chỉ định, m = m. Lý do tôi**

**không kiểm tra là sự tự chỉ định của các thành viên đã an toàn chưa: cả hai cách triển khai của tôiMa trậnnhiệm vụ**

**sao chép của sẽ hoạt động chính xác và hiệu quả hợp lý cho m = m. Ngoài ra, việc tự giao là rất hiếm, vì vậy hãy**

**kiểm tra khả năng tự giao trong một bài tập sao chép nếu bạn chắc chắn rằng mình cần.**



**Mục 17.5.1.1** **Cẩn thận với các trình xây dựng mặc định** **509**

**17.5.1.1 Cẩn thận với các trình xây dựng mặc định**

**Khi viết một thao tác sao chép, hãy đảm bảo sao chép mọi cơ sở và thành viên. Xem xét:**

**lớp X {**

**dây;**

**chuỗi s2;**

**vectơ <chuỗi> v;**

**X (const X &)**

**// *sao chép hàm tạo***

**: s {as}, v {av}**

**// *có thể cẩu thả và có thể sai***

**{**

**}**

**// *...***

**};**

* **đây, tôi '' quên '' sao chép s2, vì vậy nó được khởi tạo mặc định (để ""). Điều này không chắc là đúng. Cũng không chắc rằng tôi sẽ mắc lỗi này đối với một lớp học đơn giản. Tuy nhiên, đối với các lớp học lớn hơn, cơ hội quên sẽ tăng lên. Tệ hơn nữa, khi ai đó rất lâu sau khi thiết kế ban đầu thêm một thành viên vào một lớp, rất dễ quên thêm nó vào danh sách các thành viên được sao chép. Đây là một lý do để thích các hoạt động sao chép mặc định (do trình biên dịch tạo) (§17.6).**

**17.5.1.2 Bản sao của Căn cứ**

**Đối với mục đích sao chép, một cơ sở chỉ là một thành viên: để sao chép một đối tượng của một lớp dẫn xuất, bạn phải sao chép các cơ sở của nó. Ví dụ:**

**struct B1 {**

**B1 ();**

**B1 (const B1 &);**

* ***...***

**};**



**struct B2 {**

**B2 (int);**

**B2 (const B2 &);**

* ***...***

**};**

**struct D: B1, B2 {**

**D (int i): B1 {}, B2 {i}, m1 {}, m2 {2∗tôi} {}**

**D (const D & a): B1 {a}, B2 {a}, m1 {a.m1}, m2 {a.m2} {}**

**B1 m1;**

**B2 m2;**

**};**

**D d {1}; //*xây dựng với đối số int*D đ**

**{d}; //*sao chép cấu trúc***

**Thứ tự khởi tạo là thông thường (cơ sở trước thành viên), nhưng để sao chép thứ tự tốt hơn thì không thành vấn đề.**



**510** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**MỘT ảo base (§21.3.5) có thể xuất hiện dưới dạng cơ sở của một số lớp trong một hệ thống phân cấp. Một phương thức khởi tạo sao chép mặc định (§17.6) sẽ sao chép nó một cách chính xác. Nếu bạn xác định hàm tạo bản sao của riêng mình, kỹ thuật đơn giản nhất là sao chép liên tụcảo cơ sở. Trong đó đối tượng cơ sở là nhỏ vàảo cơ sở chỉ xảy ra một vài lần trong một hệ thống phân cấp, điều này có thể hiệu quả hơn các kỹ thuật tránh các bản sao được tái tạo.**

**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 hoạt động 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.**

**Xem xét một cảnh sát**

**chương trình định hướng giá trị**

**(§31.2.2), hoạt động phải đáp ứng hai tiêu chí:**

* ***Tương đương*: Saux = y, hoạt động trênNSvàynên cho kết quả tương tự. Đặc biệt, nếu==được xácđịnh cho loại của họ, chúng ta nên có x == y và f (x) == f (y) cho bất kỳ chức năng nào NS() điều đó chỉ phụ thuộc vào các giá trị của NS và y (trái ngược với việc hành vi của nó phụ thuộc vào địa chỉ của NS và y).**
* ***Sự độc lập*: Saux = y, hoạt động trênNSkhông nên hoàn toàn thay đổi trạng thái củay, đó làf**

**(x) không thay đổi giá trị của y miễn là f (x) không đề cập đến y.**

**Đây là hành vi NS và vectơ phục vụ. Các hoạt động sao chép cung cấp tính tương đương và độc lập dẫn đến mã đơn giản hơn và dễ bảo trì hơn. Điều này đáng được nêu rõ bởi vì mã vi phạm các quy tắc đơn giản này không phải là hiếm và các lập trình viên không phải lúc nào cũng nhận ra rằng những vi phạm đó là nguyên nhân gốc rễ của một số vấn đề tồi tệ hơn của họ. Bản sao cung cấp tính tương đương và độc lập là một phần của khái niệm về loại thông thường (§24.3.1).**

**Trước hết hãy xem xét yêu cầu về tính tương đương. Mọi người hiếm khi vi phạm yêu cầu này một cách cố ý, và các hoạt động sao chép mặc định không vi phạm nó; họ sao chép thành viên (§17.3.1, §17.6.2). Tuy nhiên, các thủ thuật, chẳng hạn như ý nghĩa của bản sao phụ thuộc vào '' các tùy chọn '', đôi khi xuất hiện và thường gây ra nhầm lẫn. Ngoài ra, không có gì lạ khi một đối tượng chứa các thành viên không được coi là một phần**



**công cụ phân bổ của nó bởi vì tất cả**

**Tương tự, các bộ đếm để thu thập thống kê và các giá trị được lưu trong bộ nhớ cache đôi khi không đơn giản chỉ được sao chép. Các phần '' phi giá trị '' như vậy của trạng thái đối tượng không được ảnh hưởng đến kết quả của các toán tử so sánh. Đặc biệt,x = y nên ngụ ý x == y. Hơn nữa, cắt (§17.5.1.4) có thể dẫn đến '' các bản sao '' hoạt động khác nhau và thường là một sai lầm tồi tệ.**

**Bây giờ hãy xem xét yêu cầu của tính độc lập. Hầu hết các vấn đề liên quan đến (thiếu) tính độc lập đều liên quan đến các đối tượng có chứa con trỏ. Ý nghĩa mặc định của sao chép là sao chép thành viên. Thao tác sao chép mặc định sao chép một thành viên con trỏ, nhưng không sao chép đối tượng (nếu có) mà nó trỏ tới. Ví dụ:**

**struct S {**

**NS∗ P;**

* ***một con trỏ***

**};**

**S x {new int {0}};**



|  |  |  |
| --- | --- | --- |
| **Mục 17.5.1.3** | **Ý nghĩa của Copy** | **511** |
| **void f ()** |  |  |
| **{** |  |  |
| **S y {x};** | **// *''sao chép '' x*** |  |

**∗yp = 1;**

**∗xp = 2;**

**xóa yp;**

**yp = new int {3};**

**∗xp = 4;**

* ***thay đổi y; ảnh hưởng đến x*// *thay đổi x; ảnh hưởng đến y*// *ảnh hưởng đến x và y*// *OK: thay đổi y; không ảnh hưởng đến x*// *oops: ghi vào bộ nhớ được phân bổ theo thỏa thuận***

**}**

* **đây tôi đã vi phạm quy tắc củaNS trạng thái của thông qua y. Đây là**

**Sự thay thế rõ ràng của việc sao chép trạng thái hoàn chỉnh của một đối tượng được gọi là *bản sao sâu*. Thông thường, giải pháp thay thế tốt hơn cho bản sao sâu không phải là một bản sao cạn mà là một hoạt động di chuyển, giúp giảm thiểu việc sao chép mà**

**không làm tăng thêm độ phức tạp (§3.3.2, §17.5.2).**

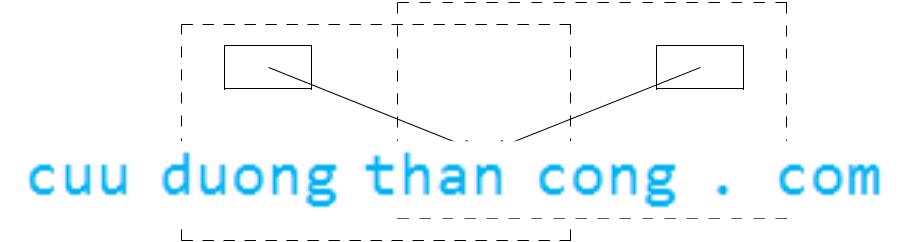
**Một bản sao cạn để lại hai đối tượng (ở đây, NS và y) với một *trạng thái chia sẻ*và có tiềm năng rất lớn**

**vì sự nhầm lẫn và sai sót. Chúng tôi nói rằng các đối tượngNS và y đã trở thành *vướng víu* khi yêu cầu độc lập đã bị vi phạm. Không thể lý luận về một đối tượng bị vướng víu một cách cô lập. Ví dụ: không rõ ràng từ mã nguồn mà hai nhiệm vụ đó∗xp có thể có những tác động khác nhau đáng kể.**

**Chúng ta có thể biểu diễn hai đối tượng vướng víu bằng đồ thị:**

**trạng thái của y:**

**trạng thái của x:**



**NS:** **y:**

**Lưu ý rằng sự vướng víu có thể phát sinh theo nhiều cách khác nhau. Thông thường, không rõ ràng là sự vướng mắc đã xảy ra cho đến khi phát sinh các thử thách. Ví dụ, một loại nhưNS có thể được sử dụng một cách thận trọng như một thành viên của một lớp có hành vi tốt khác. Tác giả ban đầu củaNS có thể nhận thức được sự vướng mắc và chuẩn bị đối phó với nó, nhưng ai đó lại cho rằng một cách ngây thơ rằng việc sao chép một NS có nghĩa là việc sao chép giá trị hoàn chỉnh của nó có thể bị ngạc nhiên và ai đó tìm thấy NS nằm sâu trong các lớp khác có thể rất ngạc nhiên.**

**Chúng tôi có thể giải quyết các vấn đề liên quan đến vòng đời của một subobject được chia sẻ bằng cách giới thiệu một hình thức thu gom rác. Ví dụ:**

**struct S2 {**

**shared\_ptr <int> p;**

**};**

**S2 x {new int {0}};**



**512** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**void f ()**

**{**

**S2 y {x};**

**∗yp = 1;**

**∗xp = 2;**

**ypreset (new int {3});**

**∗xp = 4;**

* ***''sao chép '' x***
* ***thay đổi y, ảnh hưởng đến x***
* ***thay đổi x; ảnh hưởng đến y***
* ***thay đổi y; ảnh hưởng đến x***
* ***thay đổi x; ảnh hưởng đến y***

**}**

**Trên thực tế, sao chép cạn và các vật thể vướng víu như vậy là một trong những nguồn cung cấp nhu cầu thu**

**gom rác. Vướng mắc phản đối**

**thu gom rác (ví dụ: sha**

**Tuy nhiên, một shared\_ptr vẫn là một con trỏ, vì vậy chúng ta không thể coi các đối tượng chứa shared\_ptr đang cách ly. Ai có thể cập nhật đối tượng trỏ đến? Thế nào? Khi nào? Nếu chúng ta đang chạy trong một hệ thống đa luồng, thì có cần đồng bộ hóa để truy cập vào dữ liệu được chia sẻ không? Làm thế nào chúng tôi có thể chắc chắn? Các đối tượng bị vướng (ở đây, do một bản sao cạn) là một nguồn phức tạp và các lỗi có thể được giải quyết một phần tốt nhất bằng cách thu gom rác (dưới mọi hình thức).**

**Lưu ý rằng trạng thái chia sẻ không thay đổi không phải là vấn đề. Trừ khi chúng ta so sánh các địa chỉ, chúng ta không thể biết liệu hai giá trị bằng nhau có được biểu diễn dưới dạng một hoặc hai bản sao hay không. Đây là một quan sát hữu ích vì nhiều bản sao không bao giờ được sửa đổi. Ví dụ, các đối tượng được truyền bởi giá trị hiếm khi được ghi vào. Quan sát này dẫn đến khái niệm*copy-on-write*. Ý tưởng là một bản sao thực sự không cần sự độc lập cho đến khi trạng thái được chia sẻ được ghi vào, vì vậy chúng ta có thể trì hoãn việc sao chép trạng thái được chia sẻ cho đến ngay trước khi lần đầu tiên ghi vào nó. Xem xét:**

**Hình ảnh lớp {**

**công cộng:**

* ***...***

**Hình ảnh (const Image & a);** **// *sao chép hàm tạo***

* ***...***



**void write\_block (Des**

* ***...***

**riêng:**

**Đại diện∗ dòng vô tính();**

**Đại diện∗ trả lời;**

**bool chia sẻ;**

**};**

* ***sao chép \* đại diện***

**Giả sử rằng một Đại diện có thể rất lớn và đó là một write\_block () là tốn kém so với thử nghiệm một bool. Sau đó, tùy thuộc vào việc sử dụngHình ảnhs, có thể hợp lý khi triển khai hàm tạo bản sao như một bản sao cạn:**

**Hình ảnh :: Hình ảnh (const Image & a)** **// *sao chép nông và chuẩn bị sao chép trên viết***

* **rep {a.rep},**

**đã chia sẻ {true}**

**{**

**}**

**Chúng tôi bảo vệ đối số cho phương thức khởi tạo sao chép đó bằng cách sao chép Đại diện trước khi viết:**



|  |  |  |
| --- | --- | --- |
| **Mục 17.5.1.3** | **Ý nghĩa của Copy** | **513** |
| **void write\_block (Bộ mô tả d) {** |  |  |
| **nếu (được chia sẻ) {** |  |  |
| **rep = clone ();** | **// *tạo một bản sao của \* đại*** |  |
| **chia sẻ = sai;** | ***diện*// *không chia sẻ nữa*** |  |

**}**

* ***... bây giờ chúng ta có thể an toàn viết vào bản sao đại diện của riêng mình ...***

**}**

**Giống như bất kỳ kỹ thuật nào khác, copy-on-write không phải là thuốc chữa bách bệnh, nhưng nó có thể là sự kết hợp hiệu quả giữa tính đơn giản của bản sao thật và tính hiệu quả của bản sao cạn.**

**17.5.1.4 Cắt lát**

**Một con trỏ đến một lớp dẫn xuất chuyển đổi hoàn toàn thành một con trỏ tới lớp cơ sở công khai của nó. Khi áp dụng cho hoạt động sao chép, quy tắc đơn giản và cần thiết này (§3.2.4, §20.2) dẫn đến một cái bẫy cho những người không cẩn thận. Xem xét:**

**cơ sở struct {**

**int b;**

**Cơ sở (const Cơ sở &);**

* ***...***

**};**

**struct Derived: Base {**

**int d;**

**Có nguồn gốc (const Derived**

**&); //*...***

**};**



**void ngây thơ (Cơ sở∗ P) {**

**B b2 = ∗P;** **// *có thể***

* ***...***

**}**

**void user ()**

**{**

**Có nguồn gốc d;**

**ngây thơ (& d);**

**Cơ sở bb = d;** **// *slice: gọi Base :: Base (const Base &), không Derived :: Derived (const Derived &)***

* ***...***

**}**

**Các biến b2 và bb chứa các bản sao của Cơ sở một phần của NS, tức là, một bản sao của db. Các thành viênddkhông được sao chép. Hiện tượng này được gọi là*cắt lát*. Nó có thể là chính xác những gì bạn dự định (ví dụ: xem hàm tạo bản sao cho NS trong §17.5.1.2 nơi chúng tôi chuyển thông tin đã chọn đến một lớp cơ sở), nhưng thông thường đó là một lỗi nhỏ. Nếu bạn không muốn cắt lát, bạn có hai công cụ chính để ngăn chặn điều đó:**



**514** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

1. **Cấm sao chép lớp cơ sở: xóa bỏ các hoạt động sao chép (§17.6.4).**
2. **Ngăn chặn việc chuyển đổi một con trỏ thành một con trỏ dẫn xuất thành một con trỏ thành một cơ sở: làm cho lớp cơ sở trở thành**

**riêng hoặc được bảo vệ cơ sở (§20.5).**

**Trước đây sẽ thực hiện khởi tạo b2 và bb những sai sót; cái sau sẽ thực hiện cuộc gọi của ngây thơ() và việc khởi tạo bb các lỗi.**

**17.5.2 Di chuyển**

**Cách truyền thống để nhận giá trị từ Một đến NS là sao chép nó. Đối với một số nguyên trong bộ nhớ 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, từ**

**thực hiện tráo đổi() Ví dụ**

**mẫu <lớp T>**

**void swap (T & a, T & b) {**

**const T tmp = a;**

**a = b;**

**b = tmp;**

**// *đặt một bản sao của một vào tmp*//**

***đặt một bản sao của b vào một*// *đặt***

***một bản sao của tmp vào b***

**};**

**Sau khi khởi tạo tmp, chúng tôi có hai bản sao của Mộtgiá trị của. Sau khi giao chotmp, chúng tôi có hai bản sao của NSgiá trị của. Sau khi giao choNS, chúng tôi có hai bản sao của tmpgiá trị của (nghĩa là, giá trị ban đầu của Một). Sau đó, chúng tôi phá hủytmp. Điều đó nghe có vẻ như rất nhiều công việc, và nó có thể được. Ví dụ:**

**void f (string & s1, string & s2,**

**vector <string> & vs1, vector <string> & vs2,**

**Matrix & m1, Matrix & m2)**

**{**



**hoán đổi (s1, s2);**

**hoán đổi (vs1.vs2);**

**hoán đổi (m1, m2);**

**}**

**Chuyện gì xảy ra nếu s1 có một nghìn ký tự? Chuyện gì xảy ra nếuvs2 có một nghìn phần tử mỗi một nghìn ký tự? Chuyện gì xảy ra nếum1 là 1000∗1000 ma trận của képNS? Chi phí sao chép các cấu trúc dữ liệu đó có thể là đáng kể. Trên thực tế, thư viện tiêu chuẩntráo đổi() đã luôn được thiết kế cẩn thận để tránh chi phí như vậy cho dây và vectơ. Đó là, nỗ lực đã được thực hiện để tránh sao chép (lợi dụng thực tế làdây và vectơ các đối tượng thực sự chỉ là các xử lý đối với các phần tử của chúng). Công việc tương tự phải được thực hiện để tránh một vấn đề hiệu suất nghiêm trọng đối vớitráo đổi() của Ma trậnes. Nếu thao tác duy nhất mà chúng ta có là sao chép, thì công việc tương tự phải được thực hiện đối với một số lượng lớn các hàm và cấu trúc dữ liệu không phải là một phần của tiêu chuẩn.**

**Vấn đề cơ bản là chúng tôi thực sự không muốn sao chép gì cả: chúng tôi 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 bạn thành bản sao của riêng bạn. Nếu tôi cho bạn mượn xe của tôi, tôi sẽ đưa cho bạn một chìa khóa và**



**Mục 17.5.2** **Di chuyển** **515**

**bạn lái xe đi trong xe của tôi, thay vì trong bản sao mới tạo của bạn trên xe của tôi. Một khi tôi đã cho bạn một đối tượng, bạn có nó và tôi không còn làm nữa. Do đó, chúng ta nói về '' cho đi '', '' chuyển giao '', ''**

**'chuyển quyền sở hữu' 'và' 'di chuyển' 'các đối tượng vật chất. Nhiều đối tượng trong máy tính giống với các đối tượng vật lý (mà chúng ta không sao chép mà không cần đến và chỉ với chi phí đáng kể) hơn các giá trị số nguyên (chúng ta thường sao chép vì điều đó dễ dàng và rẻ hơn so với các lựa chọn thay thế). Ví dụ như ổ khóa, ổ cắm, tay cầm tệp, luồng, chuỗi dài và vectơ lớn.**

**Để cho phép người dùng tránh các vấn đề logic và hiệu suất khi sao chép, C ++ hỗ trợ trực tiếp khái niệm *động* cũng như khái niệm về *sao chép*. Đặc biệt, chúng ta có thể xác định*di chuyển các* *nhà xây dựng* và *chuyển bài tập* để di chuyển thay vì sao chép đối số của họ. Hãy xem xét lại haichiều đơn giảnMa trận từ §17.5.1:**

**mẫu <lớp T>**

**ma trận lớp {**

**std :: array <int, 2> dim;**

**NS∗ độ cao; //*con trỏ tới các phần tử sz kiểu T***

**Ma trận (int d1, int d2): dim {d1, d2}, elem {new T [d1∗d2]} {}**

**int size () const {return dim [0]∗mờ [1]; }**

**Ma trận (const Matrix &);**

**// *sao chép hàm tạo***

**Ma trận (Matrix &&);**

**// *di chuyển hàm tạo***

**Ma trận & operator = (const Matrix &); Ma trận & operator = (Ma trận &&);**

**// *sao chép bài tập***

**// *chuyển nhiệm vụ***

**˜Matrix (); //*người phá hủy***

* ***...***

**};**

**Các && chỉ ra một tham chiếu rvalue (§7.7.2).**



* **tưởng đằng sau việc di chuyển và sao chép các giá trị của**

**hàm tạo. Cho mộttrở lại giá trị,**

**Chúng tôi có thể xác định Ma trậnPhương thức khởi tạo di chuyển để chỉ cần lấy biểu diễn từ nguồn của nó và thay thế nó bằng một giá trị trống Ma trận (mà rẻ để phá hủy). Ví dụ:**

**mẫu <lớp T>**

**Ma trận <T> :: Ma trận (Ma trận && a)**

**// *di chuyển hàm tạo***

**: dim {a.dim}, elem {a.elem}**

**// *lấy đại diện của một***

**{**

**a.dim = {0,0};**

**// *xóa đại diện của a***

**a.elem = nullptr;**

**}**

**Đối với nhiệm vụ di chuyển, chúng ta chỉ có thể thực hiện hoán đổi. Ý tưởng đằng sau việc sử dụng hoán đổi để thực hiện chuyển nhượng là nguồn sắp bị hủy, vì vậy chúng tôi có thể để bộ hủy cho nguồn thực hiện công việc dọn dẹp cần thiết cho chúng tôi:**



**516** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**mẫu <lớp T>**

**Ma trận <T> & Ma trận <T> :: operator = (Matrix && a) {** **// *chuyển nhiệm vụ***

**hoán đổi (dim, a.dim);** **// *hoán đổi đại diện***

**hoán đổi (elem, a.elem);**

**trở lại ∗cái này;**

**}**

**Di chuyển các hàm tạo và di chuyển các phép gán không cóhăng sô (rvalue) đối số tham chiếu: họ có thể và thường làm, ghi vào đối số của họ. Tuy nhiên, đối số của một hoạt động di chuyển phải luôn được để ở trạng thái mà trình hủy có thể đối phó (và tốt nhất là xử lý rất rẻ và dễ dàng).**

**Đối với các xử lý tài nguyên, m**

**các thao tác sao chép. Đặc biệt, các hoạt động di chuyển thường không ném ra các ngoại lệ; họ không có được tài nguyên hoặc thực hiện các hoạt động phức tạp, vì vậy họ không cần phải làm như vậy. Về điểm này, chúng khác với nhiều thao tác sao chép (§17.5).**

**Làm thế nào để trình biên dịch biết khi nào nó có thể sử dụng một hoạt động di chuyển thay vì một hoạt động sao chép? Trong một số trường hợp, chẳng hạn như đối với giá trị trả về, các quy tắc ngôn ngữ nói rằng nó có thể (vì hành động tiếp theo được xác định để hủy phần tử). Tuy nhiên, nói chung chúng ta phải nói nó bằng cách đưa ra một đối số tham chiếu rvalue. Ví dụ:**

**mẫu <lớp T>**

**void swap (T & a, T & b) {** **// *"hoán đổi hoàn hảo "(gần như)***

**T tmp = std :: move (a);**

**a = std :: move (b);**

**b = std :: move (tmp);**

**}**

**Các di chuyển() là một hàm thư viện chuẩn trả về một tham chiếu rvalue cho đối số của nó (§35.5.1):di chuyển (x) có nghĩa là '' thay vào đó hãy cho tôi một tên, nó cho phép người dùng đặt tên di chuyển() đã được sử dụng cho hoạt động này trong nhiều năm.**



**Các thùng chứa thư viện tiêu chuẩn có các hoạt động di chuyển (§3.3.2, §35.5.1) và do đó có các loại thư viện tiêu chuẩn khác, chẳng hạn như đôi (§5.4.3, §34.2.4.1) và unique\_ptr (§5.2.1, §34.3.1). Hơn nữa, các hoạt động chèn các phần tử mới vào các vùng chứa thư viện tiêu chuẩn, chẳng hạn nhưchèn() và push\_back (), có các phiên bản lấy tham chiếu rvalue (§7.7.2). Kết quả thực là các vùng chứa và thuật toán tiêu chuẩn mang lại hiệu suất tốt hơn những gì chúng có thể làm được nếu chúng phải sao chép.**

**Điều gì sẽ xảy ra nếu chúng ta cố gắng hoán đổi các đối tượng của một kiểu không có hàm tạo chuyển động? Chúng tôi sao chép và trả giá. Nói chung, một lập trình viên có trách nhiệm tránh sao chép quá nhiều. Công việc của trình biên dịch không phải là quyết định điều gì là thừa và điều gì là cần thiết. Để có được sự tối ưu hóa sao chép để di chuyển cho các cấu trúc dữ liệu của riêng bạn, bạn phải cung cấp các thao tác di chuyển (rõ ràng hoặc ẩn ý; xem §17.6).**

**Các loại tích hợp, chẳng hạn như NS và kép∗, được coi là có các hoạt động di chuyển mà chỉ cần sao chép.**

**Như thường lệ, bạn phải cẩn thận về cấu trúc dữ liệu có chứa con trỏ (§3.3.1). Đặc biệt, đừng cho rằng con trỏ chuyển đến được đặt thànhnullptr.**



**Mục 17.5.2** **Di chuyển** **517**

**Có các hoạt động di chuyển ảnh hưởng đến thành ngữ để trả về các đối tượng lớn từ các hàm. Xem xét:**

**Toán tử ma trận + (const Matrix & a, const Matrix & b)**

* ***res [i] [j] = a [i] [j] + b [i] [j] cho mỗi tôi và j***

**{**

**if (a.dim [0]! = b.dim [0] || a.dim [1]! = b.dim [1])**

**ném std :: runtime\_error ("kích thước Ma trận không bằng nhau trong +");**

**Ma trận res {a.dim [0], a.dim**

**[1]}; constexpr auto n = a.size ();**

**for (int i = 0; i! = n; ++ i)**

**res.elem [i] = ae**

**trả lại res;**

**}**

**Ma trận có hàm tạo di chuyển để '' trả về theo giá trị '' đơn giản và hiệu quả cũng như '' tự nhiên ''. Nếu không có hoạt động di chuyển, chúng ta sẽ gặp vấn đề về hiệu suất và phải dùng đến các giải pháp thay thế. Chúng tôi có thể đã xem xét:**

**Ma trận & toán tử + (const Matrix & a, const Matrix & b) {**

* ***hãy coi chừng!***

**Ma trận & res = ∗Ma trận mới; //*res [i]*// *phân[j]=* *bổ trên cửa hàng miễn phí***

***a [i] [j] + b [i] [j] cho mỗi tôi và j*trả lại res;**

**}**

**Việc sử dụng Mới ở trong toán tử + () không rõ ràng và buộc người dùng + để đối phó với các vấn đề phức tạp về quản lý bộ nhớ:**

* **Làm thế nào để đối tượng được tạo ra bởi Mới hiểu được xóa bỏNS?**
* **Chúng ta có cần người thu gom rác không?**
* **Chúng ta có nên sử dụng một nhóm Ma trậnthay vì chung chung Mới?**
* **Chúng ta có cần sử dụng-cou không**
* **Chúng ta có nên thiết kế lại không**
* **Phải người gọi op**
* **Điều gì xảy ra với bộ nhớ mới được cấp phát nếu tính toán ném ra một ngoại lệ?**



**Không có lựa chọn thay thế nào là thanh lịch hoặc chung chung.**

**17.6 Tạo hoạt động mặc định**

**Việc viết các hoạt động thông thường, chẳng hạn như một bản sao và một trình hủy, có thể tẻ nhạt và dễ xảy ra lỗi, vì vậy trình biên dịch có thể tạo chúng cho chúng tôi khi cần thiết. Theo mặc định, một lớp cung cấp:**

* **Một hàm tạo mặc định: NS()**
* **Một phương thức tạo bản sao: X (const X &)**
* **Một bản sao chuyển nhượng: X & operator = (const X &)**
* **Một phương thức khởi tạo di chuyển: X (X &&)**
* **Chuyển nhượng: X & operator = (X &&)**

**• Một trình hủy: ˜NS()**



**518** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**Theo mặc định, trình biên dịch tạo ra từng hoạt động này nếu một chương trình sử dụng nó. Tuy nhiên, nếu lập trình viên nắm quyền kiểm soát bằng cách xác định một hoặc nhiều hoạt động đó, thì việc tạo ra các hoạt động liên quan sẽ bị loại bỏ:**

* **Nếu người lập trình khai báo bất kỳ hàm tạo nào cho một lớp, thì hàm tạo mặc định không được tạo cho lớp đó.**
* **Nếu người lập trình khai báo một thao tác sao chép, một thao tác di chuyển hoặc một hàm hủy cho một lớp, thì không có thao tác sao chép, thao tác di chuyển hoặc hủy nào được tạo cho lớp đó.**

**Thật không may, quy tắc thứ hai chỉ được thực thi không hoàn toàn: để tương thích ngược, các trình tạo bản sao và các phép gán sao chép được tạo ngay cả khi trình hủy được xác định. Tuy nhiên, thế hệ đó không được chấp nhận trong tiêu chuẩn ISO (§iso.D) và bạn nên mong đợi một trình biên dịch hiện đại cảnh báo chống lại nó.**

**Nếu cần, chúng ta có thể được e**

**(§17.6.4).**

**17.6.1 Mặc định rõ ràng**

**Vì việc tạo ra các hoạt động mặc định khác có thể bị loại bỏ, nên phải có một cách để lấy lại mặc định. Ngoài ra, một số người muốn xem danh sách đầy đủ các thao tác trong văn bản chương trình ngay cả khi danh sách đầy đủ đó không cần thiết. Ví dụ, chúng ta có thể viết:**

**lớp gslice {**

**kích thước <size\_t> valarray;**

**valarray <kích thước\_t> sải bước;**

**valarray <kích thước\_t> d1;**

**công cộng:**

**gslice () = mặc định;**

**˜gslice () = mặc định;**

**gslice (const gslice &) = default;**

**gslice (gslice &&) = default; gslice**



* **operator = (co gslice & operator = (gs**

**// *...***

**};**

**Đoạn này của việc triển khai std :: gslice (§40.5.6) tương đương với:**

**lớp gslice {**

**kích thước <size\_t> valarray;**

**valarray <kích thước\_t> sải bước;**

**valarray <kích thước\_t> d1;**

**công cộng:**

* ***...***

**};**

**Tôi thích cái sau hơn, nhưng tôi có thể thấy điểm của việc sử dụng cái trước trong cơ sở mã được duy trì bởi các lập trình viên C ++ ít kinh nghiệm hơn: những gì bạn không thấy, bạn có thể quên.**

**Sử dụng =vỡ nợ luôn tốt hơn so với việc tự viết triển khai ngữ nghĩa mặc định của riêng bạn. Một người nào đó giả định rằng tốt hơn là nên viết một cái gì đó, thay vì không có gì, có thể viết:**



**Mục 17.6.1** **Mặc định rõ ràng** **519**

**lớp gslice {**

**kích thước <size\_t> valarray;**

**valarray <kích thước\_t> sải bước;**

**valarray <kích thước\_t> d1;**

**công cộng:**

* ***...***

**gslice (const gslice & a);**

**};**

**gslice :: gslice (const gslice & a)**

* **kích thước {a.size}, sải bước {a.stride},**

**d1 {a.d1}**

**{**

**}**

**Điều này không chỉ dài dòng, khiến việc đọc định nghĩa của gslice, nhưng cũng mở ra cơ hội cho việc mắc sai lầm. Ví dụ: tôi có thể quên sao chép một trong các thành viên và đặt nó được khởi tạo mặc định (thay vì sao chép). Ngoài ra, khi người dùng cung cấp một chức năng, trình biên dịch không còn biết ngữ nghĩa của chức năng đó và một số tối ưu hóa trở nên bị hạn chế. Đối với các hoạt động mặc định, những tối ưu hóa đó có thể rất quan trọng.**

**17.6.2 Hoạt động mặc định**

* **nghĩa mặc định của mỗi hoạt động được tạo, như được triển khai khi trình biên dịch tạo ra nó, là áp dụng hoạt động cho từng cơ sở và khôngtĩnh thành viên dữ liệu của lớp. Đó là, chúng tôi nhận được bản sao của thành viên, bản dựng mặc định của thành viên, v.v. Ví dụ:**

**struct S {**

**chuỗi a;**



**int b;**

**};**

**S f (S đối số)**

**{**

**S s0 {};** **// *cấu trúc mặc định: {"", 0}***

**S s1 {s0}; //*sao chép xây dựng***

**s1 = arg; //*sao chép bài tập*trả**

**về s1; //*chuyển công trình***

**}**

**Việc xây dựng bản sao của s1 bản sao s0.a và s0.b. Cáctrở lại của s1 di chuyển s1.a và s1.b, rời đi s1.adưới dạng chuỗi trống và s1.b không thay đổi.**

**Lưu ý rằng giá trị của một đối tượng được chuyển đến của một kiểu dựng sẵn là không thay đổi. Đó là điều đơn giản và nhanh nhất để trình biên dịch làm. Nếu chúng ta muốn một điều gì đó khác được thực hiện cho một thành viên của một lớp, chúng ta phải viết các hoạt động di chuyển của chúng ta cho lớp đó.**

**Trạng thái di chuyển từ mặc định là trạng thái mà trình hủy mặc định và gán bản sao mặc định hoạt động chính xác. Nó không được đảm bảo (hoặc bắt buộc) rằng một hoạt động tùy ý trên một đối tượng được chuyển đến sẽ hoạt động chính xác. Nếu bạn cần đảm bảo mạnh mẽ hơn, hãy viết các hoạt động của riêng bạn.**



**520** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**17.6.3 Sử dụng Thao tác Mặc định**

**Phần này trình bày một số ví dụ chứng minh cách sao chép, di chuyển và hủy được liên kết một cách hợp lý. Nếu chúng không được liên kết, các lỗi hiển nhiên khi bạn nghĩ về chúng sẽ không được trình biên dịch bắt gặp.**

**17.6.3.1 Trình tạo mặc định**

**Xem xét:**

**struct X {**

**X (int);** **// *yêu cầu một***

**};**

**Bằng cách khai báo một hàm tạo yêu cầu một đối số nguyên, lập trình viên tuyên bố rõ ràng rằng người dùng cần cung cấp một NS để khởi tạo một NS. Nếu chúng tôi cho phép phương thức khởi tạo mặc định được tạo, quy tắc đơn giản đó sẽ bị vi phạm. Chúng ta có:**

**X a {1};** **// *VÂNG***

**X b {};** **// *lỗi: không có hàm tạo mặc định***

**Nếu chúng ta cũng muốn có hàm tạo mặc định, chúng ta có thể xác định một hoặc khai báo rằng chúng ta muốn hàm tạo mặc định do trình biên dịch tạo ra. Ví dụ:**

**cấu trúc Y {**

**dây;**

**int n;**

**Y (chuỗi const & s);** **// *khởi tạo Y bằng một chuỗi*// *cho phép khởi tạo mặc***

**Y () = mặc định;** ***định với nghĩa mặc định***

**};**

**Hàm khởi tạo mặc định mặc định (tức là, được tạo) cấu trúc mặc định mỗi thành viên. Ở đây,Y () bộ NS vào chuỗi trống. '' Khởi tạo mặc định '' của một thành viên tích hợp khiến thành viên đó chưa được khởi tạo. Thở dài! Hy vọng cho một com**



**17.6.3.2 Duy trì Bất biến**

**Thông thường, một lớp có một bất biến. Nếu vậy, chúng tôi muốn các hoạt động sao chép và di chuyển để duy trì nó và trình hủy giải phóng mọi tài nguyên liên quan. Thật không may, trình biên dịch không thể trong mọi trường hợp biết lập trình viên coi cái gì là bất biến. Hãy xem xét một ví dụ hơi xa vời:**

**struct Z {// *bất biến:***

**// *my\_favorite là chỉ mục của phần tử yêu thích của tôi trong elem*//**

***điểm lớn nhất đến phần tử có giá trị cao nhất trong elem***

**vectơ <int> elem;**

**int my\_favorite;**

**NS∗ lớn nhất;**

**};**

**Lập trình viên đã nêu một bất biến trong nhận xét, nhưng trình biên dịch không đọc nhận xét. Hơn nữa, lập trình viên đã không để lại một gợi ý về cách mà bất biến đó được thiết lập và duy trì. Đặc biệt, không có hàm tạo hoặc phép gán nào được khai báo. Bất biến đó là**



**Mục 17.6.3.2** **Duy trì Bất biến** **521**

**ngầm hiểu. Kết quả là mộtZ có thể được sao chép và di chuyển bằng các thao tác mặc định:**

**Z v0;**

**Z val {{1,2,3}, 1, & val [2]};**

**Z v2 = val;**

**// *không có khởi tạo (rất tiếc! khả năng có giá trị không xác định)*// *Được,* *nhưng xấu và dễ xảy ra lỗi*// *bản sao: v2.largest điểm vào val*// *di chuyển: val.elem trở nên trống rỗng; v3.my\_favorite nằm ngoài phạm vi***

**Z v3 = di chuyển (val);**

**Đây là một mớ hỗn độn. Vấn đề gốc rễ làZ được thiết kế tồi vì thông tin quan trọng bị '' ẩn '' trong nhận xét hoặc bị thiếu hoàn toàn. Các quy tắc để tạo ra các hoạt động mặc định là heuristic nhằm phát hiện những sai lầm phổ biến và khuyến khích một cách tiếp cận có hệ thống để xây dựng, sao chép, di chuyển và phá hủy. Bất cứ nơi nào có thể**

1. **Thiết lập một invar**
2. **Duy trì invar**
3. **Thực hiện mọi thao tác dọn dẹp cần thiết trong trình hủy (bao gồm cả giải phóng tài nguyên).**

**17.6.3.3 Tài nguyên bất biến**

**Nhiều ứng dụng quan trọng và rõ ràng nhất của bất biến liên quan đến quản lý tài nguyên. Hãy xem xét một đơn giảnXử lý:**

**mẫu <class T> class Xử lý {**

**NS∗ P;**

**công cộng:**

**Xử lý (T∗ pp): p {pp} {} T &**

**toán tử∗() { trở lại ∗P; }**

**˜Handle () {xóa p; }**

**};**

* **tưởng là bạn xây dựng một Xử lý đã cho một con trỏ đến một đối tượng được cấp phát bằng cách sử dụng Mới. CácXử lý cung cấp quyền truy cập vào đối tượng được trỏ đến và cuối cùng xóa bỏs đối tượng đó. Ví dụ:**

**void f1 ()**



**{**

**Xử lý <int> h {new**

* ***...***

**}**

**Xử lý khai báo một phương thức khởi tạo nhận đối số: điều này ngăn chặn việc tạo ra phương thức khởi tạo**

**mặc định. Điều đó tốt vì một hàm tạo mặc định có thể để lạiXử lý <T> :: p chưa khởi tạo:**

**void f2 ()**

**{**

**Xử lý <int> h; //*lỗi: không có hàm tạo mặc định*//**

***...***

**}**

**Sự vắng mặt của một hàm tạo mặc định giúp chúng ta tránh khỏi khả năng xóa bỏ với một địa chỉ bộ nhớ ngẫu nhiên.**

**Cũng, Xử lý khai báo một trình hủy: điều này ngăn chặn việc tạo ra các hoạt động sao chép và di chuyển. Một lần nữa, điều đó cứu chúng ta khỏi một vấn đề khó chịu. Xem xét:**



**522**

**Xây dựng, Dọn dẹp, Sao chép và Di chuyển**

**Chương 17**

**void f3 ()**

**{**

**Xử lý <int> h1 {new int {7}}; Xử lý <int> h2 {h1};**

**// *lỗi: không có hàm tạo bản sao***

**// *...***

**}**

**Có Xử lý có một hàm tạo bản sao mặc định, cả hai h1 và h2 sẽ có một bản sao của con trỏ và cả hai sẽ có xóa bỏd nó. Kết quả sẽ không xác định và rất có thể là thảm họa (§3.3.1). Lưu ý: việc tạo ra các hoạt động sao chép chỉ không được chấp nhận, không bị cấm, vì vậy nếu bạn bỏ qua các cảnh báo, bạn có thể nhận được exa này**

**sao chép mặc định và di chuyển quyền sở hữu ope sents, memberw và memberwise copy *Là* thích hợp, rõ ràng =vỡ nợ và một nhận xét rất có thể là một ý kiến hay.**

**Nếu chúng tôi muốn xây dựng bản sao, chúng tôi có thể xác định một cái gì đó như:**

**mẫu <lớp T>**

**lớp học Xử lý {**

* ***...***

**Xử lý (const T & a): p {new T {∗ap}} {}** **// *dòng vô tính***

**};**

**17.6.3.4 Bất biến được chỉ định một phần**

**Các ví dụ rắc rối dựa vào các bất biến nhưng chỉ thể hiện một phần chúng thông qua các hàm tạo hoặc hàm hủy là hiếm hơn nhưng không phải là chưa từng thấy. Xem xét:**

**lớp Tic\_tac\_toe {**

**công cộng:**

**Tic\_tac\_toe (): pos (9) {} // *luôn là 9 vị trí***



**Tic\_tac\_toe & operat**

**{**

**for (int i = 0; i <9; ++ i)**

**pos.at (i) = arg.pos.at (i); trở**

**lại∗cái này;**

**}**

* ***... các hoạt động khác ...***

**enum State {trống, không, chéo};**

**riêng:**

**vectơ <Trạng thái> pos;**

**};**

**Điều này đã được báo cáo là một phần của một chương trình thực tế. Nó sử dụng '' số ma thuật ''9 để triển khai một phép gán bản sao truy cập đối số của nó tranh luận mà không cần kiểm tra xem đối số có thực sự có chín phần tử hay không. Ngoài ra, nó thực hiện việc gán bản sao một cách rõ ràng, nhưng không thực hiện phương thức tạo bản sao. Đây không phải là những gì tôi coi là mã tốt.**



**Mục 17.6.3.4** **Những người bất biến được chỉ định một phần** **523**

**Chúng tôi đã xác định phép gán bản sao, vì vậy chúng tôi cũng phải xác định hàm hủy. Bộ hủy đó có thể là**

* **mặc định bởi vì tất cả những gì nó cần làm là đảm bảo rằng thành viên pos bị phá hủy, đó là những gì sẽ được thực hiện nếu không xác định việc gán bản sao. Tại thời điểm này, chúng tôi nhận thấy rằng bài tập sao chép do người dùng xác định về cơ bản là bài tập mà chúng tôi sẽ nhận theo mặc định, vì vậy chúng tôi có thể=vỡ nợ điều đó cũng có. Thêm một hàm tạo bản sao để có sự hoàn chỉnh và chúng tôi nhận được:**

**lớp Tic\_tac\_toe {**

**công cộng:**

**Tic\_tac\_toe (): pos (9) {} // *luôn là 9 vị trí***

**Tic\_tac\_toe (const Tic\_tac\_toe &) = default;**

**Tic\_tac\_toe & operator = (const Tic\_tac\_toe & arg) = default;**

**˜Tic\_tac\_toe () = defa**

* ***... các hoạt động khác ...***

**enum State {trống, không, chéo};**

**riêng:**

**vectơ <Trạng thái> pos;**

**};**

**Nhìn vào điều này, chúng tôi nhận thấy rằng hiệu quả ròng của những =vỡ nợs chỉ để loại bỏ các hoạt động di chuyển. Đó là những gì chúng ta muốn? Chắc là không. Khi chúng tôi thực hiện nhiệm vụ sao chép=vỡ nợ, chúng tôi đã loại bỏ sự phụ thuộc khó chịu vào hằng số ma thuật 9. Trừ khi các hoạt động khác trênTIC Tac Toe, chưa được đề cập cho đến nay, cũng được '' kết nối với các con số ma thuật '', chúng ta có thể thêm các thao tác di chuyển một cách an toàn. Cách đơn giản nhất để làm điều đó là xóa=vỡ nợs, và sau đó chúng tôi thấy rằng TIC Tac Toe thực sự là một loại hoàn toàn bình thường:**

**lớp Tic\_tac\_toe {**

**công cộng:**

* ***... các hoạt động khác ...***



**enum State {trống, không, chéo};**

**riêng:**

**vectơ <Trạng thái> pos {V**

**};**

**Một kết luận mà tôi rút ra từ điều này và các ví dụ khác trong đó '' kết hợp kỳ lạ '' của các hoạt động mặc định được xác định là chúng ta nên rất nghi ngờ về các kiểu như vậy: sự bất thường của chúng thường che giấu các sai sót trong thiết kế. Đối với mọi lớp, chúng ta nên hỏi:**

1. **Có cần một hàm tạo mặc định không (vì hàm mặc định không đủ hoặc đã bị chặn bởi một hàm tạo khác)?**
2. **Có cần thiết bị hủy không (ví dụ: vì một số tài nguyên cần được giải phóng)?**
3. **Các thao tác sao chép có cần thiết không (vì ngữ nghĩa sao chép mặc định không đầy đủ, ví dụ: vì lớp được hiểu là một lớp cơ sở hoặc vì nó chứa các con trỏ đến các đối tượng mà lớp phải xóa)?**
4. **Các thao tác di chuyển có cần thiết không (vì ngữ nghĩa mặc định không đầy đủ, ví dụ: vì một đối tượng trống không có ý nghĩa)?**

**Đặc biệt, chúng ta không bao giờ nên chỉ xem xét một trong những hoạt động này một cách riêng lẻ.**



**524** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

**17.6.4 xóa bỏd Chức năng**

**Chúng ta có thể '' xóa '' một hàm; nghĩa là, chúng ta có thể nói rằng một hàm không tồn tại để cố gắng sử dụng nó (một cách ngầm hiểu hoặc rõ ràng) là một lỗi. Cách sử dụng rõ ràng nhất là loại bỏ các chức năng mặc định khác. Ví dụ, người ta thường muốn ngăn chặn việc sao chép các lớp được sử dụng làm cơ sở vì việc sao chép như vậy dễ dẫn đến cắt (§17.5.1.4):**

**lớp Cơ sở {**

* ***...***

**Cơ sở & operator = (const Base &) = delete; // *không cho phép sao***

***chép*Cơ sở (const Base &) = xóa;**

**Cơ số & toán tử = (Ba**

**Cơ sở (Base &&) = xóa;**

**};**

**Căn số x1;**

**Cơ sở x2 {x1}; //*lỗi: không có hàm tạo bản sao***

**Việc bật và tắt tính năng sao chép và di chuyển thường được thực hiện thuận tiện hơn bằng cách nói những gì chúng tôi muốn (sử dụng =vỡ nợ; §17.6.1) thay vì nói những gì chúng tôi không muốn (sử dụng=xóa bỏ). Tuy nhiên, chúng ta có thểxóa bỏ bất kỳ chức năng nào mà chúng ta có thể khai báo. Ví dụ: chúng ta có thể loại bỏ một chuyên môn khỏi tập hợp các chuyên môn có thể có của một mẫu hàm:**

**mẫu <lớp T>**

**NS∗ nhân bản (T∗ P) // *trả lại bản sao của \* p*{**

**trả lại T {∗P};**

**};**

**Foo∗ nhân bản (Foo∗) = xóa;** **// *đừng cố gắng sao chép một Foo***



**void f (Hình dạng∗ ps, Foo∗ pf {**

**Hình dạng∗ ps2 = bản sao** **// *fine***

**(ps); Foo∗ pf2 = clone (pf);** **// *error: clone (Foo \*) đã bị xóa***

**}**

**Một ứng dụng khác là loại bỏ một chuyển đổi không mong muốn. Ví dụ:**

**cấu trúc Z {**

**// *...***

**Z (gấp đôi);** **// *có thể khởi tạo với một đôi*// *nhưng***

**Z (int) = xóa;** ***không phải với một số nguyên***

**};**

**void f ()**

**{**

**Z z1 {1};** **// *lỗi: Z (int) đã bị xóa*//**

**Z z2 {1,0};** ***VÂNG***

**}**



**Mục 17.6.4** **xóa bỏd Chức năng** **525**

**Một cách sử dụng nữa là kiểm soát nơi có thể cấp phát một lớp:**

**lớp Not\_on\_stack {**

* ***...***

**˜Not\_on\_stack () = xóa;**

**};**

**class Not\_on\_free\_store {**

* ***...***

**vô hiệu∗ toán tử new (size\_t) = xóa;**

**};**

**Bạn không thể có một phiên bản địa**

**phương trong cửa hàng miễn phí khi bạn**

**yêu cầu:**

**void f ()**

**{**

**Not\_on\_stack v1;**

**Not\_on\_free\_store v2;**

**// *lỗi: không thể phá hủy*//**

***VÂNG***

**Not\_on\_stack∗ p1 = new Not\_on\_stack;** **// *VÂNG***

**Not\_on\_free\_store∗ p2 = new Not\_on\_free\_store;** **// *lỗi: không thể phân bổ***

**}**

**Tuy nhiên, chúng ta không bao giờ có thể xóa bỏ điều đó Not\_on\_stack Kỹ thuật thay thế để làm cho sự vật. người phá hủyriêng (§17.2.2) có thể giải quyết vấn đề đó.**

**Lưu ý sự khác biệt giữa =xóa bỏd và một hàm chưa được khai báo. Trong trường hợp trước đây, trình biên dịch lưu ý rằng lập trình viên đã cố gắng sử dụngxóa bỏd chức năng và đưa ra một lỗi. Trong trường hợp thứ hai, trình biên dịch tìm kiếm các lựa chọn thay thế, chẳng hạn như không gọi hàm hủy hoặc sử dụng toàn cụctoán tử mới ().**



**17.7 Lời khuyên**

1. **Thiết kế hàm tạo, phép gán và hàm hủy dưới dạng một tập hợp các thao tác; §17.1.**
2. **Sử dụng một hàm tạo để thiết lập một bất biến cho một lớp; §17.2.1.**
3. **Nếu một hàm tạo có được một tài nguyên, thì lớp của nó cần một hàm hủy để giải phóng tài nguyên đó;**

**§17.2.2.**

1. **Nếu một lớp có một hàm ảo, nó cần một hàm hủy ảo; §17.2.5.**
2. **Nếu một lớp không có hàm tạo, nó có thể được khởi tạo bằng cách khởi tạo thành viên; §17.3.1.**
3. **Thích {} khởi tạo hơn = và () sự khởi tạo; §17.3.2.**
4. **Cung cấp cho một lớp một hàm tạo mặc định nếu và chỉ khi có giá trị mặc định '' tự nhiên ''; §17.3.3.**
5. **Nếu một lớp là một vùng chứa, hãy cung cấp cho nó một phương thức khởi tạo danh sách khởi tạo; §17.3.4.**
6. **Khởi tạo các thành viên và cơ sở theo thứ tự khai báo của chúng; §17.4.1.**
7. **Nếu một lớp có một thành viên tham chiếu, nó có thể cần các hoạt động sao chép (hàm tạo sao chép và gán sao chép); §17.4.1.1.**



**526** **Xây dựng, Dọn dẹp, Sao chép và Di chuyển** **Chương 17**

1. **Thích khởi tạo thành viên hơn là gán trong một phương thức khởi tạo; §17.4.1.1.**
2. **Sử dụng bộ khởi tạo trong lớp để cung cấp các giá trị mặc định; §17.4.4.**
3. **Nếu một lớp là một xử lý tài nguyên, nó có thể cần các hoạt động sao chép và di chuyển; §17.5.**
4. **Khi viết một phương thức khởi tạo sao chép, hãy cẩn thận sao chép mọi phần tử cần được sao chép (cẩn thận với các trình khởi tạo mặc định); §17.5.1.1.**
5. **Hoạt động sao chép phải cung cấp tính tương đương và độc lập; §17.5.1.3.**
6. **Cẩn thận với các cấu trúc dữ liệu vướng víu; §17.5.1.3.**
7. **Thích chuyển ngữ nghĩa và copy-on-write sang copy cạn; §17.5.1.3.**
8. **Nếu một lớp được sử dụng làm lớp cơ sở, hãy bảo vệ khỏi việc cắt lát; §17.5.1.4.**
9. **Nếu một lớp cần một hoạt động sao chép hoặc một hàm hủy, nó có thể cần một hàm tạo, một hàm hủy, một phép gán bản sao, một**
10. **Nếu một lớp có một tions**

**điểm; §17.6.3.3.**

**[21] Nếu một lớp là một xử lý tài nguyên, nó cần một hàm tạo, một hàm hủy và các hoạt động sao chép không mặc định; §17.6.3.3.**

1. **Nếu một hàm tạo, phép gán hoặc hàm hủy mặc định phù hợp, hãy để trình biên dịch tạo ra nó (đừng tự viết lại); §17.6.**
2. **Nói rõ ràng về những bất biến của bạn; sử dụng các hàm tạo để thiết lập chúng và các phép gán để duy trì chúng; §17.6.3.2.**
3. **Đảm bảo rằng các bài tập sao chép là an toàn để tự chuyển nhượng; §17.5.1.**
4. **Khi thêm một thành viên mới vào một lớp, hãy kiểm tra xem có các hàm tạo do người dùng định nghĩa cần được cập**

**nhật để khởi tạo thành viên đó hay không; §17.5.1.**

