DỊCH SÁCH THE C++ PROGRAMMING LANGUAGE

CHƯƠNG 16

Các lớp

Những loại đó không phải là “trừu tượng”;

Chúng thực như int và float

-Doug Mcllroy

* Giới thiệu
* Kiến thức cơ bản về kinh điển
* Chức năng thành viên; sao chép mặc định; kiểm soát truy cập;lớp và cấu trúc; người xây dựng; người xây dựng rõ ràng; bộ khởi tạo trong lớp; định nghĩa hàm trong lớp ; tính đột biến; tự giới thiệu; quyền truy cập thành viên; thành viên tĩnh; 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 bê tông các lớp học

* 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 khuôn mẫu (＄3.4, chương 23) cho phép người lập trình biểu diễn 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 float tích hợp trong C++ với các phép toán +, -,\*,v.v.., cung cấp một giá trị gần đúng cụ thể về khái niệm toán học của 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 loại Trunk\_line trong chương trình xử lí điện thoại, loại Explosion cho trò chơi điện tử hoặc loại listeParagraph> cho chương trình xử lí văn bản. Một chương trình cung cấp các kiểu phù hợp chặt chẽ với các khái niệm của ứng dụng có xu hướng dễ hiểu hơn, dễ lý luận hơn và dễ sửa đổi hơn một chương trình không được lựa chọn tốt về các kiểu do người dùng xác định cũng làm cho 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.

Ý tưởng cơ bản trong việc xác định một kiểu mới alf tách các chi tiết ngẫu nhiên của quá trình triễn khai ( ví dụ như bố cục của dữ liệu được sử dụng để lưu trữ một đối tượng của kiểu) khỏi các thuộc tính cơ bản để sử dụng nó một cách chính xác ( ví dụ: danh sách đầy đủ các chức năng có thể truy cập dữ liệu) . Sự riêng 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” do người dùng xác định mà về mặt logic không khác nhiều so với các kiểu cài sẵn:

S16.2 Cơ bản về lớp giới thiệu các phương tiện cơ bản để xác định một lớp và các thành viên của nó.

S6.3. Lớp Conerete thảo luận về thiết kế của lớp bê tông thanh lịch và thành thạo.

Các chương trình tiếp theo đi vào chi tiết hơn và trình bày cấu trúc phân cấp Classes và Elasses trừu tượng

Chương 1 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 copy 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: đi ra khỏi phạm vi).

Chương 18;:Nạp chồng toán tử giải thích cách xác định toán tử đơn phân và nhị phân( chẳng hạn như + và 1) cho các kiểu do người dùng định nghĩa và cách sử dụng chúng.

Chương 19 Các toán tử đặc biệt xem xét các xác định và sử dụng các toán tử ( chẳng hạn như IO> new) "đặc biệt ở chỗ chúng thường được sử dụng theo cách rút gọn từ các toán tử số học và logic. Đặc biệt, chương này trình bày cách xác định một chuỗi elass . Chương 20 Các lớp có nguồn gốc giới thiệu các tính năng cơ bản của ngôn ngữ hỗ trợ lập trình theo đối tượng-ori. Các lớp cơ sở và dẫn xuất, các hàm ảo và bộ điều khiển truy cập được bao phủ. Chương 21 Phân cấp lớp tập trung vào việc sử dụng các lớp cơ sở và lớp dẫn xuất để tổ chức mã một cách khéo léo xung quanh khái niệm phân cấp lớp. Phần lớn chương này được dành để thảo luận về các kỹ thuật lập trình, nhưng các khía cạnh kỹ thuật của tính kế thừa đa (các lớp có nhiều hơn một lớp cơ sở) cũng được đề cập. t uesDes the Techniques điều hướng rõ ràng các cấu trúc phân cấp lớp. Đặc biệt, các hoạt động chuyển đổi kiểu dynamic\_cast và static\_cast là presenteu, như 1 là toán hạng l0r xác định kiểu ne hoặc ooJect cho một trong các lớp cơ sở của nó (typeld). 16.2 Cơ bản về Class Dưới đây là một bản tóm tắt rất ngắn gọn về các lớp: Một mệnh đề 18 một kiểu người dùng-dennea. Một lớp bao gồm một tập hợp các thành viên. Các loại thành viên phổ biến nhất là ghi nhớ dữ liệu- Ders and ember Tunctions. Các hàm thành viên có thể xác định ý nghĩa ot initalization (tạo), coPY, di chuyển và dọn dẹp (phá hủy).

Các thành viên được truy cập bằng cách sử dụng. (dấu chấm) cho các đối tượng và -> (mũi tên) cho con trỏ. Các toán tử, chẳng hạn như +, 1 và [1. có thể được định nghĩa cho một lớp. Một lớp là một không gian tên chứa các thành viên của nó. Các thành viên công khai cung cấp giao diện của lớp và các thành viên riêng cung cấp thông tin chi tiết về mentauon. Một cấu trúc là một lớp mà các thành viên là công khai theo mặc định. Đối với lớp exampic. SX {private: đại diện (triển khai) là prvate public: user intertace là pubic X (int i = 0) m) {} một hàm tạo (khởi tạo thành viên dữ liệu m) W một thành viên funcion nt mf (int i) d = m; l đặt một giá trị mới trả về cũ; trả lại giá trị cũ X var {7}; Biến Ila kiểu X, được khởi tạo thành 7 int user (X var, X ° ptr) int x = var.mf () nty = pirn nevar access using. (dấu chấm) truy cập uSg> (arow oror cannor acce55 private member Các phần sau mở rộng về điều này và đưa ra cơ sở lý luận. Phong cách là hướng dẫn: phát triển dần dần các ý tưởng, với các chi tiết được hoãn lại cho đến sau này. 16.2.1 Chức năng thành viên Xem xét triển khai khái niệm ngày bằng cách sử dụng struct ($ 2.3.1, $ 8.2) để xác định đại diện của Ngày và một tập hợp các hàm để thao tác với các biến kiểu này: struct Date W biểu diễn int d, m, Yi void init date (Date & d, int, int, int) initializeP void add year (Date & d, int n); void add\_month (Ngày & d, int n); a0d n năm để thêm n tháng vào quảng cáo void add\_day (Date & d, int n); Thêm n ngày vào Không có kết nối rõ ràng nào giữa kiểu dữ liệu, Ngày tháng và các hàm này. Một kết nối như vậy có thể được thiết lập bằng cách khai báo các Chức năng là thành viên:

struct Date (int d, m, Y vold init (int dd, int mm, int yy); Initialz2 n năm W thêm n tháng vold thêm năm (int n) void thêm tháng (int n); void add\_day (int n)}; W thêm n dayss Các hàm được khai báo trong định nghĩa lớp (struct là một loại lớp; $ 16.2.4) được gọi là các hàm thành viên và chỉ có thể được gọi cho một biến cụ thể của kiểu thích hợp bằng cách sử dụng cú pháp tiêu chuẩn để truy cập thành viên cấu trúc (S8.2). Ví dụ: Ngày sinh nhật của tôi; void T0 Ngày hôm nay; today.init16,10,1996): mybirthday.init (30,12,1950); Ngày mai = hôm nay; ngày mai.add ngày (); Vì các cấu trúc khác nhau có thể có các hàm thành viên trùng tên, chúng ta phải chỉ định tên cấu trúc khi định nghĩa một hàm thành viên: void Date: init (int dd, int mm, int yy) d dd 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 đó Trường hợp, tên đề cập đến thành viên của đối tượng mà hàm được gọi. Đối với kỳ thi, khi Ngày: init) i1s được gọi cho ngày hôm nay, m = mm gán cho ngày hôm nay.m. Mặt khác, khi Ngày: init () được gọi cho my\_birthday, m = mm gán cho my\_ birthday.m. Một hàm thành viên lớp "biết nó được gọi cho đối tượng nào. Nhưng hãy xem §16.2.12 để biết khái niệm về thành viên tĩnh. 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 ngày sinh của tôi: khởi tạo bằng bản sao Khởi tạo ngày dz (my\_birthday) bằng bản sao

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 đối với lớp X, một hành vi thích hợp hơn có thể được cung cấp (Ș3.3, Ș17.5). Tương tự như vậy, các đối tượng của lớp có thể được sao chép bởi phép gán. Ví dụ: void f (Date & 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 X, người dùng có thể xác định một toán tử gán thích hợp ($ 3,3, S17,5). 16.2.3 Kiểm soát truy cập Khai báo Ngày trong tiểu mục trước pr Cung cấp một tập hợp các hàm để thao tác Ngày. Tuy nhiên, nó không chỉ rõ rằng những hàm đó phải là những hàm duy nhất phụ thuộc trực tiếp vào biểu diễn của Date và những hàm duy nhất để truy cập trực tiếp vào các đối tượng có thể được thể hiện bằng cách sử dụng một lớp thay thế hoặc một lớp cấu trúc. Ngày công khai: void startnt dd, int mm, int yy); I khởi tạo l add n yeh void add\_year (int n); void add davint ay (un n addn dav ay Nhãn công khai phân tách nội dung lớp thành hai phần. Các tên trong phần đầu tiên, riêng tư, chỉ có thể được sử dụng bởi các hàm thành viên. Phần thứ hai, công khai, tạo thành giao diện công khai đối với objectS của lớp. Một cấu trúc đơn giản là một lớp có các thành viên là công khai theo mặc định (S16.2.4); các chức năng thành viên có thể được định nghĩa và sử dụng chính xác như trước đây. Ví dụ: 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: lỗi void timewarp (Date & d): Date :: y là private 200 Hàm init) bây giờ rất cần thiết vì việc đặt dữ liệu ở chế độ riêng tư buộc chúng ta phải cung cấp cách khởi tạo các thành viên. Ví dụ Ngày dx dx.m = 3; dx.init (25,3,2011); W error: m là riêng tư CHẢO

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 khiến Ngày 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 gỡ lỗi - bản địa hóa - đã hoàn thành cho dù chương trình thậm chí còn được chạy. Đây là một bài phát biểu- Trường hợp quan trọng của quan sát chung rằng bất kỳ thay đổi nào đối với hành vi của loại Ngày có thể và phải được thực hiện bởi các thay đổi đối với các thành viên của nó, Đặc biệt, nếu chúng ta thay đổi đại diện của một lớp, chúng ta chỉ cần thay đổi các chức năng của thành viên để tận dụng đại diện 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 bữa ăn trưa dành cho thành viên để học cách sử dụng một lớp học. Một lợi thế nhỏ hơn, nhưng quan trọng nhất, là tập trung vào thiết kế một giao diện tốt chỉ đơn giản là dẫn đến mã tốt hơn bởi vì những suy nghĩ và ime olherwISC dành cho việc gỡ lỗi được dành cho những mối quan tâm liên quan đến việc sử dụng đúng cách. Việc bảo vệ đối với dữ liệu riêng tư dựa trên sự hạn chế hoặc việc sử dụng hoặc tên các thành viên trong lớp. Do đó, nó có thể bị phá vỡ bằng thao tác địa chỉ ($ 7,4.) 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 ý gian lận (gian lận). Chỉ phần cứng mới có thể cung cấp sự bảo vệ hoàn hảo chống lại việc sử dụng độc hại một lan truyền mục đích chung, và thậm chí điều đó khó có thể thực hiện được trong các hệ thống thực tế. 16.2.4 class và struct ne construct class X (..) được gọi là định nghĩa lớp; nó định nghĩa một kiểu gọi là x. Vì lý do lịch sử, một định nghĩa lớp thường được coi là một khai báo lớp. Ngoài ra, giống như các khai báo không phải là định nghĩa, một phân định lớp có thể được sao chép trong các nguồn khác nhau bằng cách sử dụng #include mà không vi phạm quy tắc phân tách một lần ($ 15,2,3). Theo định nghĩa, struct là một lớp trong đó các thành viên được mặc định là public; nghĩa là, struct s (. "7}% Đơn giản chỉ cần viết tắt cho lớp S (public: "..")}; Hai định nghĩa này của S có thể thay thế cho nhau, mặc dù thông thường sẽ khôn ngoan hơn nếu bạn chỉ theo một kiểu. Phong cách nào bạn sử dụng tùy thuộc vào hoàn cảnh và sở thích. Tôi có xu hướng sử dụng cấu trúc cho các lớp mà tôi nghĩ là cấu trúc dữ liệu Just Smple. nghĩ ot một lớp là ype thích hợp với một lớp sử dụng, bất biến. Các hàm tạo và hàm truy cập có thể khá hữu ích ngay cả đối với các cấu trúc, nhưng như một cách viết tắt chứ không phải là người bảo đảm cho các bất biến (S $ 24,3,2, S13,4). Theo detault, các thành viên của một lớp là private: class Date1 (, m, Yi private bởi detauit public: Date1 (int dd, int thêm n năm void add year (int n); thêm n năm

Tuy nhiên, chúng tôi cũng có thể sử dụng công cụ xác định quyền truy cập riêng tư: để nói rằng các thành viên sau đây là riêng tư, Cũng giống như công khai: nói rằng các thành viên sau đây là publhc: struct Date2 (private: Int d, m, Yi public Date2 (int dd, int mm, int yy); void thêm năm (int n); tôi thêm n năm Ngoại trừ tên khác, Date1 và Date2 là tương đương nhau. Không bắt buộc phải khai báo dữ liệu trước trong một lớp. Trên thực tế, thường hợp lý khi đặt các thành viên dữ liệu cuối cùng để nhấn mạnh các chức năng cung cấp giao diện người dùng công khai. Ví dụ: class Date3 (public: Ngày (int dd, int mm, int yy); void add\_year (int n); Tôi thêm n năm riêng tư: int d, r Trong mã thực, nơi mà 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: class Date4 public Date4 int dd, int mm, int yy) i private nt d, m, Yi public: void add year (int n); thêm n năm Tuy nhiên, có nhiều hơn một phần công khai, như trong Date4, sẽ gây 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 Bộ xây dựng Việc sử dụng các hàm như init) để cung cấp khởi tạo cho các đối tượng lớp là không phù hợp và dễ xảy ra lỗi. Bởi vì không có chỗ nào nói rằng một đối tượng phải được khởi tạo, một lập trình viên có thể quên làm như vậy hoặc làm như vậy hai lần (thường cho kết quả thảm hại như nhau). Một cách tiếp cận tốt hơn là cho phép pro0 grammer khai báo một hàm với mục đích rõ ràng là khởi tạo các đối tượng. Bởi vì một hàm như vậy xây dựng các giá trị của một kiểu nhất định, nó được gọi là một hàm tạo. Một phương thức khởi tạo được công nhận bởi có cùng tên với chính lớp đó. Ví dụ:

lớp Ngày (d, m, yi public: Ngày (int dd, int mm, int 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); viết tắt oeir Ngày xmas (25,12,1990); eet 0 (10,12): Ngày phát hành1\_0 (10,12);: thiếu đối số thứ i rO 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); dạng viết tắt Lỗi ngày dropasel\_0 (10,12): thiếu đối số thứ ba Tôi khuyên bạn nên sử dụng ký hiệu {) trên ký hiệu 0 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 (S2.2.2, $ 6.3.5). Có những trường hợp phải sử dụng ký hiệu 0 (94.4.1, S[17.3.2.1](https://l.facebook.com/l.php?u=http://17.3.2.1/?fbclid=IwAR3K110ftiALlnq7r0uUat34ymrXumLWz9urq4nAn34DQiBR78pLeym8mGc&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank)), 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 ype. Ví dụ: class Date int d, m, Y public: Ngày (int, nt); day, monin, day day, montn, todays yoar Ngày W, tháng và năm hôm nay / Ngày mặc định: ngày hôm nay trong biểu diễn chuỗi nt, Ngày (int, n Ngày (int); Ngày(); Ngày (const char-); Các hàm tạo tuân theo các quy tắc nạp chồng giống như các hàm thông thường ($ 12,3). Miễn là các cấu trúc có đủ sự khác biệt về kiểu đối số của chúng, trình biên dịch có thể chọn một cấu trúc phù hợp để sử dụng: Ngày hôm nay (4) uly 4, 1983 "); Ngày bây giờ W 4, today.m, todayy 15, 11, o oav fefault inil mặc định được khởi tạo như hôm nay Ngày bắt đầu (}; Sự gia tăng của các hàm tạo trong ví dụ Ngày là điển hình. Khi thiết kế một lớp học, người lập trình 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 o

giảm số lượng các hàm liên quan là sử dụng các đối số mặc định ($ 12,2,5). Đối với Ngày, mỗi đối số có thể được cung cấp một giá trị mặc định được hiểu là "chọn giá trị mặc định: hôm nay". lớp Ngày int d, m, y; công cộng: Ngày (int dd = 0, int mm -0, int yy = 0); Date :: Date (int da, int mm, înt yy) as dd T dd: today.d, m = mm? mm: hôm nay.m; Y = yy 7 yy: today.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 bộ giá trị có thể có cho đối số. Ngày và tháng của tuần, điều này rõ ràng là như vậy, nhưng trong 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 IBC (năm = m-1). Ngoài ra, chúng ta có thể sử dụng các giá trị mặc định trực tiếp làm đối số mặc định: class Date Ngày (int dd = today.d, int mm = today.m, int yy = today-.y); Ngày :: Ngày (int dd, int mm, int yy) 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 vào khoảng thời gian của Date. 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 (S16.3.1). 16.2.6 Trình tạo rõ ràng 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: W da = (1.0) (55.6.2) gấp đôi> d {1

Những chuyển đổi ngầm như vậy có thể cực kỳ hữu ích. Số phức là một ví dụ: chúng ta loại bỏ phần ảo, chúng ta 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 xét Ngày: void my\_ tct (Ngày d); 10 tuổi Ngày d (15)}; hợp lý: x trở thành (15, today.m.today.y) my\_Ict (15); mờ mịt ODscur "**\*** Tốt nhất, điều này là tối nghĩa. Không có mối liên hệ logic rõ ràng nào giữa số 15 và Ngày độc lập với sự phức tạp của mã của chúng ta. May mắn thay, chúng ta có thể chỉ định rằng một hàm tạo không được sử dụng như một chuyển đổi ngầm định. Một cấu trúc hoặc được khai báo với từ khóa rõ ràng chỉ có thể được sử dụng để khởi tạo và chuyển đổi rõ ràng. ví dụ: hoặc class Date f 1, m, Yi public: explicit Date (int dd = 0, int mm = 0, int yy = 0); Ngày d1 (15 l OK: được coi là rõ ràng Ngày d2 = Ngày (15); Ngày d3 = {15); W HOẶC: lỗi expWCi: nuanizaton không thực hiện chuyển đổi ẩn Ngày d4 = W eror: = initalization không thực hiện chuyển đổi ngầm định 15; void 10 my\_fct (15) Lỗi W: truyền đối số không thực hiện chuyển đổi ngầm định của tôi ((15}}; eror: truyền đối số không thực hiện chuyển đổi ngầm định ct của tôi (Ngày {15); OK: Một khởi tạo với dấu = được coi là một khởi tạo sao chép. 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 một thao tác di chuyển ($ 5,3,2, S17,5,2) có thể được sử dụng nó mà bộ khởi tạo là một rvalue ($ 6,4,1). Bỏ dấu = làm cho quá trình khởi tạo trở nên 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 số rõ ràng. Bạn cần một lý do chính đáng để không làm như vậy (đối với phức tạp). Nếu bạn xác định một phương thức khởi tạo ngầm định, thì tốt nhất là bạn nên ghi lại lý do của mình hoặc người bảo trì có thể nghi ngờ rằng bạn là 1orgettul (hoặc không biết gì).

Nếu một hàm tạo được khai báo rõ ràng và được định nghĩa bên ngoài lớp, thì không thể lặp lại hàm tạo đó rõ ràng: class Date (int d, m, Yi public: explicit Date (int dd); .. Ngày :: Ngày (int dd) {P .. 7} Ngày rõ ràng: Ngày (int dd) {.. 7} lỗi HOẶC Hầu hết các ví dụ trong đó rõ ràng là quan trọng liên quan đến một đối số phương thức khởi tạo duy nhất. Tuy nhiên, tường minh cũng có thể hữu ích cho các hàm tạo có 0 hoặc nhiều hơn một đối số. Ví dụ: struct X { NS(); licit Xint, intjs P X x1 = (}: Xx2 = {1,2); eror: ẩn eror: mpicit K x3 ( X x4 {1,2); OK: rõ ràng Tôi OK: explicit int 1 (X); int i eror mpucit eror: ipicit l OK: explicit int i3 = f (X ()); nt 14 f (X (1,2)); OK: rõ ràng Sự khác biệt giữa khởi tạo trực tiếp và sao chép được duy trì cho quá trình khởi tạo danh sách (ST7. [7.3.4.3](https://l.facebook.com/l.php?u=http://7.3.4.3/?fbclid=IwAR2gyJmf_5_8CCSPghuG3DwvkfDCaQ0r1KE4uLAWF8dqQwi8VXMS2gbUTEg&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank)). 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: alass Date f int d, m, Y \* public: Ngày (int, int, int); Ngày (int, int); W ngày, tháng, năm W ngày, tháng, năm hôm nay Ngày (int) Ngày(); Ngày (const char-); W ngày, ngày tháng và năm mặc định Ngày: ngày hôm nay trong biểu diễn chuỗi ..

Chúng ta có thể giải quyết vấn đề đó bằng cách giới thiệu các đối số mặc định để giảm số lượng hàm tạo (S16.2.5). Ngoài ra, chúng ta có thể thêm các trình khởi tạo vào các thành viên dữ liệu: class Date int d (today.d); int m (today.m; int y (today-y}; public Ngày (int, int, int; Ngày (int, i int) Ngày (int); ngày, tháng, năm # dav tháng, ngày của năm hôm nay, tháng và năm hôm nay M detault Date: ngày hôm nay trong biểu diễn chuỗi Ngày) Ngày (const char-); Bây giờ, mỗi hàm tạo có d, m và y được khởi tạo trừ khi chính nó thực hiện nó. Ví dụ: Date :: Date (int dd) d {dd) kiểm tra xem Ngày có hợp lệ không Điều này tương đương với: Ngày: Ngày (int dd) ddd), mtoday.m), y (hôm nay.y) để kiểm tra xem Ngày có hợp lệ không. 16.2.8 Định nghĩa hàm trong lớp Một hàm thành viên được định nghĩa trong định nghĩa lớp thay vì chỉ được khai báo ở đó - được coi là một hàm thành viên nội tuyến (§12.1.5). Có nghĩa là, định nghĩa trong lớp của các hàm thành viên dành cho các hàm nhỏ, hiếm khi được sửa đổi, được sử dụng thường xuyên. Giống như định nghĩa lớp mà nó là một phần, một hàm thành viên được phân định trong lớp có thể được sao chép trong một số đơn vị dịch bằng cách sử dụng #include. Giống như lớp iEsel, ý nghĩa của thành viên Iunction phải giống nhau trong đóverit là #included (9 15.2..3). Một thành viên có thể quan hệ với một thành viên khác nếu không có tên tuổi của nó hoặc khi thành viên đó 1 được xác định (90.3.4). Hãy xem xét: class Date (public: vold add month (int n) {m + = n;} ncrement the Dates m private: int d, m, y Nghĩa là, các khai báo thành viên chức năng và dữ liệu là độc lập với thứ tự. Tương đương tôi có thể đã viết

lớp Ngày công khai: void add month (intn) {m + = n;) mcrement the Dates m private: int d, m, yi inline void Date :: add\_month (int n) M add n months m + = n; Tôi tăng ngày của m 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 Ngày: thêm tháng; chỉ thêm n và hy vọng đạt được một ngày tốt đẹp là quá ngây thơ (16,3,1 đô la). 16.2.9 Tính đột biến Chúng ta có thể định nghĩa một đối tượng được đặt tên là một hằng số hoặc một biến. Nói cách khác, tên có thể tham chiếu đến một đối tượng chứa một giá trị không thể thay đổi hoặc có thể thay đổi. Vì thuật ngữ chính xác có thể hơi vụng về, chúng tôi kết thúc việc đề cập đến một số biến là hằng số hoặc ngắn gọn vẫn là biến const. Tuy nhiên, điều này nghe có vẻ kỳ lạ đối với người nói tiếng Anh bản ngữ, nhưng khái niệm này rất hữu ích và được nhúng 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 tìm thấy một cách cẩn thận và đôi khi cải thiện hiệu suất. Đặc biệt, tính bất biến 18 một thuộc tính hữu ích nhất trong chương trình đa luồng ($ .3, Chương 4). lo be usetul ngoài định nghĩa ot các hằng số đơn giản hay các kiểu buill-in, chúng ta phải có khả năng định nghĩa các hàm hoạt động trên các đối tượng const của kiểu do người dùng định nghĩa. Đối với các hàm tự do có nghĩa là các hàm nhận const T & đối số, Đối với các lớp, điều đó có nghĩa là chúng ta phải có khả năng xác định các hàm thành viên hoạt động trên các đối tượng const. [16.2.9.1](https://l.facebook.com/l.php?u=http://16.2.9.1/?fbclid=IwAR3ueOL6Y_PSyiBtEvsFrDKU1lcEN3f4trdGOwZplAjQOcETA1vmQTSjNYE&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank) Các chức năng thành viên không đổi Ngày như được định nghĩa cho đến nay cung cấp các hàm thành viên để cung cấp một giá trị cho Ngày. Rất tiếc, chúng tôi không cung cấp cách kiểm tra giá trị của Ngày. Sự cố này có thể dễ dàng được khắc phục bằng cách thêm các chức năng đọc ngày, tháng và năm: Ngày (lớp int d, i public: int day) const (return d;) int month) const {return m; } nt năm) Const; void add\_year (int n); thêm n năm

Hằng số sau danh sách đối số (trống) trong khai báo hàm chỉ ra rằng các hàm này không thay 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: int Date :: year) const return + y error: cố gắng thay đổi giá trị thành viên trong hàm const) Khi một hàm thành viên const được định nghĩa bên ngoài lớp của nó, thì const suftix là bắt buộc: int Ngày: năm) eor: const bị thiếu trong kiểu hàm thành viên return y; Nói cách khác, const là một phần của kiểu Date:: day (), Date :: month) và Date: year (). Một hàm thành viên const có thể được gọi cho cả đối tượng const và không phải const, trong khi một hàm thành viên không phải const chỉ có thể được gọi cho các đối tượng không phải const. Ví dụ: void f (Date & d, const Date & cd) int ayear d.add\_year (1); VÂNG W OK int j = cd.year); OK cd.add\_year (1); eror: không thể thay đổi giá trị của hằng số Ngày [16.2.9.2](https://l.facebook.com/l.php?u=http://16.2.9.2/?fbclid=IwAR29vIrwzdIYIE-2SxInwMP2itGEIg74SD2aEKAfW9OG3PRT9d2ocH-okH8&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank) Hằng số vật lý và logic Đôi khi, một hàm thành viên là hằng số về mặt logic, nhưng nó vẫn cần thay đổi giá trị của một ghi nhớ. Có nghĩa là, đối với người dùng, hàm dường như không thay đổi trạng thái của đối tượng của nó, nhưng một số chi tiết mà người dùng không thể quan sát trực tiếp được cập nhật. Điều này thường được gọi là hằng số logic. Ví dụ, lớp Ngày có thể có một hàm trả về biểu diễn chuỗi. Xây dựng đại diện này có thể là một hoạt động tương đối tốn kém. Do đó, sẽ rất hợp lý khi giữ một bản sao để các yêu cầu lặp đi lặp lại sẽ chỉ trả lại bản sao, trừ khi giá trị của Ngày 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 hãy xem nó có thể đạt được như thế nào đối với Date: class Date public: .. string string\_rep () const; đại diện chuỗi riêng tư: Dool cache\_valld; bộ nhớ cache chuỗi; vold compute cache\_value): "cache

Từ quan điểm của người dùng, string\_ rep không thay đổi trạng thái Ngày của nó, vì vậy rõ ràng nó phải là một hàm thành viên const. Mặt khác, các thành viên cache và cache\_valid phải thay đổi theo cách riêng 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 đúc, ví dụ, một ép kiểu const (SI1.5.2). 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](https://l.facebook.com/l.php?u=http://16.2.9.3/?fbclid=IwAR10rWwH0NjMMDGfjZp5WiuaNNUDVKquvFaHL71nIV9Oo9awSMRFoGyliZI&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank) có thể thay đổi Chúng ta có thể định nghĩa một thành viên của một lớp là có thể thay đổi, nghĩa là nó có thể được sửa đổi ngay cả trong một đối tượng const: class Date string stringrep () const; riêng: W chuỗi đại diện cho bộ đệm bool có thể thay đổi hợp lệ; bộ đệm chuỗi có thể thay đổi; vold compute cache\_value) consi; 7 (có thể thay đổi) bộ nhớ cache Bây giờ chúng ta có thể định nghĩa string\_rep () theo cách rõ ràng: string Ngày: string\_ repO const if (lcache\_valid) {pucache val ue0; Cacne\_vala rue trả về bộ nhớ cache; Bây giờ chúng ta có thể sử dụng chuỗi đại diện) cho cả đối tượng const và không phải const. Ví dụ: void f (Date d, const Date cd) string s1 d.stringrep) string s2 cd.string\_rep (): [16.2.9.4](https://l.facebook.com/l.php?u=http://16.2.9.4/?fbclid=IwAR3Oi5johMFLg7WFCh-EE4fSIpZ-Rn3w5CIKesc12lEsBYGO1VCncOZgam0&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank) Tính đột biến thông qua Hướng dẫn Khai báo một thành viên có thể thay đổi là thích hợp nhất khi chỉ một phần nhỏ của 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 sử dụng kỹ thuật hat, ví dụ string- with-cache sẽ trở thành:

Struet ache bool valld string rep; Ciass Chuỗi ngày string\_rep) const; private: chuỗi đại diện / khởi tạo trong hàm tạo che c; void compute\_cache\_value () const; fiW bộ nhớ cache nào tham chiếu đến chuỗi Ngày :: string\_ rep) const if (le-> valld) {compute cache\_value (0: -> 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 const 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 như vậy 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 tham khảo Các hàm cập nhật trạng thái add\_year (), add month) và add\_day0 (S16.2.3) được xác định không 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: vold f (Date & d) d.add day1) .add\_month (1) .add\_year (1); để thêm một ngày, một tháng và một năm vào d. Để làm điều này, mỗi hàm phải được khai báo để trả về một tham chiếu đến một Date: class Date

Ngày & thêm năm (int n); thêm n năm Ngày & thêm tháng (int n); // thêm n tháng Ngày & add\_ day (int n); thêm n ngày Mỗi hàm thành viên (không tĩnh) biết đối tượng nào mà nó được gọi và có thể tham chiếu đến nó một cách rõ ràng. Ví dụ: Ngày & Ngày: add\_year (int n if (dm = 29 && m = 2 8 & leapyear (y + n) {/ cẩn thận vào ngày 29 tháng 2 ngày 3 tháng này; trở lại Biểu thức này đề cập đến đối tượng mà một hàm thành viên được gọi. Trong một hàm thành viên không tĩnh, từ khóa này là một con trỏ trỏ đến đối tượng mà hàm được gọi. Trong một hàm thành viên không phải hằng số của lớp X, kiểu của hàm này là X \*. Tuy nhiên, điều này Được coi là một giá trị, Vì vậy, 1t không thể bắt anh ta giải quyết hoặc điều này hoặc để gán cho điều này. trong một hàm thành viên const của lớp X, loại 1s const X này để ngăn chặn việc sửa đổi đối tượng Bản thân nó (xem thêm $ 7,5). Hầu hết việc sử dụng điều này là ngầm định. Cụ thể, mọi liên quan đến thành viên không tĩnh Irom trong một lớp đều dựa vào việc sử dụng ngầm định điều này để có được thành viên của đối tượng thích hợp. Ví dụ, hàm add\_year có thể tương tự, nhưng thật tệ, đã được định nghĩa như thế này: Date & Date: zadd\_year (int n) if (this-> d = 29 && this-> m == 2 && tleapyear (this-> y + n)) {nis this-> m 3; của anh ấy- + n; cái này; trở lại Một cách sử dụng rõ ràng phổ biến của điều này là trong thao tác danh sách liên kết. Ví dụ: struct Lin re: Liên kết suc; dữ liệu int; Chèn liên kết (int x) insert x before this return pre = new Link {pre, this, x)

void remove) loại bỏ và hủy bỏ cái này if (pre) pre-> suc = suc supre = pre; xóa cái này đi; Cần sử dụng rõ ràng điều này để truy cập vào các thành viên của các lớp cơ sở từ một lớp dẫn xuất là một khuôn mẫu (926.3.7). 16.2.11 Quyền truy cập thành viên Một thành viên của lớp X có thể được truy cập bằng cách áp dụng. (dấu chấm) toán tử cho một đối tượng của lớp X hoặc bằng cách áp dụng toán tử thne -> (mũi tên) cho một con trỏ đến một đối tượng o lớp X. Ví dụ: struct X4 void t0 nt m void user (X x, X \* px) m1 eror: không có m trong phạm vi n = 1; l sai số: x không phải là poinie m = 1; OK px.m 1; eor: px là một con trỏ Rõ ràng, có một chút dư thừa ở đây: trình biên dịch biết liệu một tên có liên quan đến X hay X, vì vậy một toán tử duy nhất nên được sử dụng tối ưu. Tuy nhiên, một lập trình viên có thể bị 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: ĐẾN Tôi OK: "this-> m = 1;" (S16.2.10) m1 Đó là, một tên thành viên không đủ điều kiện hoạt động như thể nó đã được bắt đầu bởi this->. Lưu ý rằng một thành viên Hàm có thể tham chiếu đến tên của một thành viên mà nó đã được khai báo: struct X int Tot return m; } tine: trả về XS này m int m; Nếu chúng ta muốn liên quan đến một thành viên nói chung, thay vì một thành viên của một đối tượng cụ thể, chúng ta chấp nhận tên lớp theo sau. Ví dụ:

struct S m; int T0 stanc meS int X: f0 {return m; } nt sm int (3 :: 9) pmto (& s: IMG_256Thành viên tĩnh của W X sm ($ 16,2,12) W X's thành viên f 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 của quy tắc cho:. 16.2.12 [tĩnh) Thành viên Sự tiện lợi của giá trị mặc định cho Ngày đã được mua bằng cái giá của một vấn đề ẩn đáng kể. Ngày nay lớp Date của chúng ta trở nên phụ thuộc vào biến toàn cục. Lớp Ngày này chỉ có thể được sử dụng trong ngữ cảnh mà ngày 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 thú vị khi sử dụng các mệnh đề phụ thuộc ngữ cảnh như vậy và việc bảo trì trở nên lộn xộn. Có thể "Chỉ một biến toàn cục nhỏ 1sn t quá không thể quản lý được, nhưng stylee đó dẫn đến mã vô dụng ngoại trừ lập trình viên ban đầu của nó. Nó nên được hỗ trợ. 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 là một thành viên tĩnh. Có chính xác một bản sao của một thành viên tĩnh thay vì một bản sao cho mỗi đối tượng, như đối với các thành viên không tĩnh thông thường ($ 6,4,2). Tương tự, một hàm cần truy cập vào các thành viên của một lớp, nhưng không cần phải được gọi cho một đối tượng cụ thể, được gọi là hàm thành viên tĩnh. Đây là một thiết kế lại bảo toàn ngữ nghĩa của các giá trị phương thức khởi tạo mặc định cho Ngày mà không gặp các vấn đề bắt nguồn từ việc phụ thuộc vào một global: class Date int d, m yi 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 Ngày (dd, mm.yy) Bây giờ chúng ta có thể xác định hàm tạo Ngày để sử dụng default\_date như thế này Ngày :: Ngày (int dd, int mm, int yy dd 7 dd: default\_date.d; m = mm mm: defaultdate.m y "yyTy: detauit date.yi Tôi .. kiểm tra xem Ngày có hợp lệ không.

Sử dụng set default (), chúng ta có thể thay đổi ngày mặc định khi thích hợp. Một thành viên tĩnh có thể được gọi như bất kỳ thành viên nào khác. Ngoài ra, một thành viên tĩnh có thể được tham chiếu mà không cần đề cập đến một đối tượng. Thay vào đó, tên của nó được định nghĩa bằng tên của lớp của nó. Ví dụ: vold T0 Ngày :: set\_default (4,5,1945); call\_defauit () f thành viên tĩnh của Date được sử dụng, một thành viên tĩnh - một hàm hoặc thành viên dữ liệu - phải được định nghĩa ở đâu đó. Từ khóa static không được lặp lại trong định nghĩa của một thành viên static. Ví dụ: Ngày Date :: default\_date (16,12,1770} định nghĩa về Ngày: deauit\_ date void Date :: set\_defaultțint d, int m, int y) W định nghĩa về Date :: set\_ default date = {d, m, y); W gán giá trị mới cho ngày defauit\_ Bây giờ, giá trị mặc định là ngày sinh của Beethoven - cho đến khi ai đó quyết định khác. Lưu ý rằng Ngày {() đóng vai trò như một ký hiệu cho giá trị của Ngày :: ngày mặc định. Ví dụ: Date copy\_of default\_date = Ngày {0 oid 1 (Ngày); vô hiệu đi T (Datel) Do đó, chúng tôi không cần một chức năng riêng biệt để đọc ngày mặc định. Hơn nữa, trong đó loại đích rõ ràng là Ngày tháng, đơn giản 9 là phù hợp. Ví dụ: void fi (Date) void 12 (Date); oid 12 (int); vold go 11 (0); W OK: tương đương với f1 (Ngày (lỗi 0: không rõ ràng: 12 (nt) hoặc f2 (Ngày)? VÂNG T2 12 (Ngày (); Trong mã đa luồng, các thành viên dữ liệu tĩnh yêu cầu một số loại kỷ luật khóa hoặc truy cập để tránh các điều kiện chủng tộc (95.3.4, S41.2.4). Do đó, đa luồng hiện nay rất phổ biến, thật không may rằng việc sử dụng các thành viên dữ liệu tĩnh đã khá phổ biến trong các mã cũ hơn. Mã cũ hơn có xu hướng sử dụng các thành viên tĩnh theo cách có nghĩa là các điều kiện chủng tộc.

16.2.13 Các loại thành viên Các kiểu và bí danh kiểu có thể là thành viên của một lớp. Ví dụ: templatectypename T class Tree l bí danh thành viên sử dụng kiểu giá trị = T; Chính sách enum (rb, splay, treeps}; thành viên lớp enum Tôi thành viên lớp Nút bên phải; Nút Node ert; value\_ype value; public: void f (Cây Node top public oid g (const Ta); .. Một lớp thành viên (thường được gọi là lớp lồng nhau) có thể tham chiếu đến các kiểu và các thành viên tĩnh của lớp bao quanh nó. Nó chỉ có thể tham chiếu đến các thành viên không tĩnh 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 các ví dụ kiểu "0 và 90 thuần túy về mặt kỹ thuật, Một lớp lồng nhau có quyền truy cập vào các thành viên của lớp bao quanh nó, thậm chí đến các thành viên riêng (giống như một hàm thành viên có), nhưng không có khái niệm về đối tượng hiện tại của lớp bao quanh. Ví dụ: templatectypename T> void Tree :: Node: f (Tree p) right; lỗi hàng đầu: không có đối tượng của loại Cây được chỉ định CHẢO P-> lop ngnt value type v = leff-> value; W OK: value\_type không được liên kết với một đối tượng 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: templatectypename 1 void Tree: g (Tree :: Node p) error: no object of type Tree: Node Ve.gnt S privare value type val right-> value; value\_typev = p-> giànnt-> giá trị; p -> (cái này); 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 các kỹ thuật lập trình chung dựa trên các kiểu liên kết ($ 28,2,4, $ 33,1,3). Các enum thành viên thường là một giải pháp thay thế cho các lớp enum khi nói đến việc tránh làm ô nhiễm phạm vi bao quanh với tên của các điều tra viên ($ 8,4.).

16.3 Lớp bê tông Phần trước đã thảo luận về các bit và phần của thiết kế lớp Date trong bối cảnh giới thiệu các tính năng ngôn ngữ cơ bản để định nghĩa lớp. Ở đây, tôi đảo ngược sự nhấn mạnh và thảo luận về thiết kế của một lớp Ngày đơn giản và efiCient, đồng thời chỉ ra cách các tính năng ngôn ngữ hỗ trợ ký hiệu này 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 Toating-point, số phức, điểm. cặp pomters, coor- dinates, transtorms, pomterofser), ngày tháng, imes, phạm vi, liên kết, liên kết, nút, cặp (giá trị, đơn vị), vị trí đĩa, vị trí mã nguồn, giá trị tiền tệ, dòng, hình chữ nhật, số điểm ixed tỷ lệ , số với phân số, chuỗi ký tự, vectơ và mảng. Mỗi ứng dụng sử dụng một số ot này. Thông thường, một số ít loại bê tông đơn giản này được sử dụng nhiều. Một ứng dụng điển hình sử dụng một số trực tiếp và nhiều ứng dụng gián tiếp khác từ các thư viện. C + hỗ trợ trực tiếp một vài kiểu 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 cụ thể nhỏ phù hợp c P pes ac Cacu cụ thể pes hoặc clasSSes cụ thể để phân biệt chúng với các lớp trừu tượng ($ 20,4) và các lớp trong phân cấp lớp (S20.5, S21.2). Một lớp được gọi là cụ thể (hoặc một lớp cụ thể) nếu đại diện của nó là một phần của sự nhận biết của nó. Điều này phân biệt nó với các lớp trừu tượng ($ 5.2.2, S20.4) cung cấp một 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, Ș17,5) Để tham chiếu trực tiếp đến các đối tượng được đặt tên (thay vì truy cập thông qua con trỏ và biểu thức) Điều này làm cho các lớp cụ thể trở nên đơn giản để giải thích và dễ dàng trình biên dịch tạo ra mã tối ưu. Do đó, chúng tôi xử lý các lớp cụ thể là 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 (S4,4). Đó là một mục đích rõ ràng ban đầu của C ++ để hỗ trợ rất tốt việc sử dụng sai lệch và thành thạo đối với các kiểu do người dùng định nghĩa như vậy. Chúng là một nền tảng của lập trình tao nhã. Như thường lệ, các số 1 đơn giản và trần tục theo thống kê nhiều Signiicant hơn so với phức tạp và phức tạp. Theo cách này, chúng ta hãy xây dựng một lớp Ngày tốt hơn: không gian tên chrono enum class Tháng {jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec; class Date {public: public interface: class Bad\_date {}; / lớp ngoại lệ Ngày rõ ràng (int dd = {0, Tháng mm =}, Int yy IMG_256W0 có nghĩa là "chọn một defaulf

W các hàm không theo phương thức để kiểm tra Ngày: int day) const; Tháng tháng) const; int năm) const; / chuỗi đại diện chuỗi chuỗi đại diện) const; oa CarTOp (cnar BLI, Trong tối đa) const; Csyle stning ropresentation Các chức năng W (moditying) để thay đổi Ngày: Ngày & addyear (int n); Ngày & tháng năm (int n); Date & add\_ day (int n) i private: thêm năm, thêm n tháng W thêm ngày bool is\_valid (); kiểm tra xem Ngày này có đại diện cho ngày 7 không, m, yi bool là ngày (int d, Tháng m, int Y); đúng với ngày hợp lệ W true nó y là một bước nhảy vọt yoa bool là một bước nhảy vọt (int y) i bool operatorm = (const Date & a, const Date & b); bool operatorl = (const Ngày & a, const Ngày & b); const Ngày & default\_date |; luồng ngày phát hiện & operatoree (ostream & os, const Date & d); in d tới os istream & operator> a (istream & is, Date & d); } / Chrono Tôi đọc Ngày từ là thành một Tập hợp các thao tác này khá điển hình cho kiểu do người dùng xác định: Một hàm tạo chỉ định cách khởi tạo các đối tượng / biến của kiểu (š16,2.5). 2 Một tập hợp các chức năng cho phép người dùng kiểm tra Ngày. Các hàm này được đánh dấu là const để chỉ ra rằng chúng không sửa đổi trạng thái của đối tượng / biến mà chúng được gọi. 3] Một tập hợp các chức năng cho phép người dùng điều chỉnh Ngày tháng mà không thực sự phải biết chi tiết của biểu diễn hoặc tìm hiểu về sự phức tạp của ngữ nghĩa. 14 Các hoạt động được xác định ngầm cho phép Ngày được sao chép tự do ($ 16,2,2). S Một lớp, Bad\_date, được sử dụng để báo cáo lỗi như là ngoại lệ. 6 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 biểu diễn của Ngày, nhưng chúng được xác định là có liên quan bởi việc sử dụng không gian tên chrono. Tôi đã xác định loại Tháng để đối phó với vấn đề nhớ thứ tự tháng / ngày, chẳng hạn, để tránh nhầm lẫn về việc ngày 7 tháng 6 được viết (6,7) (kiểu Mỹ) hay (7,6) (Châu Âu Phong cách). bắt đầu giới thiệu các loại Ngày và Năm riêng biệt để đối phó với sự nhầm lẫn có thể xảy ra Ngày (1995, Tháng:; jul, 27) và Ngày (27, Tháng: jul, 1995). Tuy nhiên, các loại này sẽ không hữu ích bằng loại Tháng. Hầu như tất cả các lỗi như vậy đều bị bắt khi chạy ume- ngày 26 tháng 7 năm 2 /

không phải là một ngày phổ biến trong công việc của tôi. Xử lý các niên đại lịch sử trước năm 1800 hoặc lâu hơn là một mẹo nhỏ khó nhất dành cho các nhà sử học lão luyện. Hơn nữa, không thể kiểm tra chính xác ngày hoặc tháng đó cách ly 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 để cung cấp Lưu ý mặc định rằng đối với Tháng, số 0 cho giá trị (mặc định) 0 Cũng giống như số nguyên tor mặc dù không phải một Tháng hợp lệ ($ 8,4). Tuy nhiên, trong trường hợp này, đó chính xác là những gì chúng tôi muốn: một giá trị không hợp pháp khác để đại diện cho "chọn giá trị mặc định". Cung cấp một acrault (ví dụ: giá trị tách rời 1 hoặc Phản đối ngày) 1s ä probiem thiết kế ricky. Đối với một số Ypes, 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 như Ngày), nơi 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 - ít nhất là ban đầu - không cung cấp giá trị mặc định. Tôi chủ yếu cung cấp một cho Date để có thể thảo luận về cách làm như vậy. T 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 làm chi tiết triển khai mà không ảnh hưởng đến giao diện người dùng Đây là một ví dụ nhỏ - và nguyên bản - về cách có thể sử dụng Ngày: void f (Date & d) Ngày lvb\_ day {16, Tháng: Dec, d.year () O if (d.day () == 29 88 d.month | j == Month:: feb) (if (nửa đêm0) d.add \_day (1 ); cout "day arter:" <d + l << n; Ngày đ; W được khởi tạo vào ngày detauit cin >> dd; (ddw = d) cout <"HurrayIin"; Điều này giả định rằng toán tử bổ sung, +, đã được khai báo cho Ngày. Tôi làm điều đó trong §16.3.3. Lưu ý việc sử dụng chứng chỉ rõ ràng của dec và feb theo Tháng. Tôi đã sử dụng một lớp 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 Date (int day, month, year ;: 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 trực tiếp với các thành phần của Dates 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

gây ra công việc bổ sung cho mọi người dùng của cấu trúc. Ngoài ra, mặc dù loại Ngày có vẻ đơn giản, nhưng cần phải suy nghĩ một chút để làm đúng. Ví dụ, việc tăng Ngày phải giải quyết các năm nhuận, với khoảng thời gian mà 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 cần phải điều chỉnh ony một tập hợp được mô tả o1 Tunctions. Ví dụ: để biểu thị Ngày là số ngày trước hoặc sau ngày 1 tháng 1 năm 1970, chúng tôi cần thay đổi các Chức năng thành viên của Ngày tháng 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 (85.3.1). T đã nghiêm túc xem xét việc loại bỏ hoàn toàn khái niệm về một ngày mặc định. Điều đó sẽ buộc người dùng phải khởi tạo rõ ràng Ngày của họ một cách nhất quán, Tuy nhiên, điều đó có thể gây bất tiện và đáng ngạc nhiên, và quan trọng hơn là các giao diện phổ biến được sử dụng cho mã chung yêu cầu cấu trúc mặc định (S17.3.3). Điều đó có nghĩa là 1, với tư cách là người thiết kế Ngày, phải chọn ngày mặc định. T đã 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, việc loại bỏ set\_default\_ date) đã làm mất đi tính tổng quát của Ngày. 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 sàng lọc trong tương lai, tôi đã khai báo default date () như một hàm trợ giúp: const Date & 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. Đối với d ple: Ngày :: Ngày (int dd, Tháng mm, int yy): d {dd), m {mm), y {yy) it (y 0) y = defaultdate (.year (); it (m - = Tháng ( )) m ngày mặc định (0 tháng): f (d == 0) d = default\_date (). ngày); if (lis\_ valid0) ném Bad date (); Hàm tạo kiểm tra xem dữ liệu được cung cấp có biểu thị Ngày hợp lệ hay không. Nếu không, giả sử, đối với (30, Month :: teb, 1994), nó ném ra một ngoại lệ ($ [2.4.3.1](https://l.facebook.com/l.php?u=http://2.4.3.1/?fbclid=IwAR3MnSt8FN285SjKyOipBaJSlI2n84VacGtx2QL6qoIqgyx0r_eRtQUhoqE&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank), Chương 15), 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, 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](https://l.facebook.com/l.php?u=http://2.4.3.2/?fbclid=IwAR3Oi5johMFLg7WFCh-EE4fSIpZ-Rn3w5CIKesc12lEsBYGO1VCncOZgam0&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank), S13,4). Tôi đang sử dụng giá trị Month () - không đại diện cho một tháng và có giá trị nguyên 0- để đại diện cho "chọn tháng mặc định." Tôi có thể đã xác định một điều tra viên trong Tháng cụ thể để

đại diện cho điều đó. Nhưng tôi quyết định rằng tốt hơn nên sử dụng một giá trị rõ ràng là bất thường để biểu thị "chọn tháng mặc định" hơn là đưa ra hình thức rằng có 13 tháng trong một năm. Lưu ý rằng Tháng {}, nghĩa là 0, có thể được sử dụng vì nó nằm trong phạm vi được đảm bảo cho việc liệt kê Tháng (88,4). Tôi sử dụng cú pháp khởi tạo thành viên (S17.4) để khởi tạo các thành viên. Sau đó, tôi kiểm tra 0 và sửa đổi các giá trị nếu cần. Điều này rõ ràng không cung cấp tốc độ tối ưu trong trường hợp (hy vọng hoàn toàn hiếm) xảy ra lỗi, nhưng việc sử dụng các bộ khởi tạo thành viên sẽ làm mất đi cấu trúc của mã. Điều này làm cho phong cách ít bị lỗi hơn và bảo trì lâu 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 coi việc đặt chức năng xác nhận là hợp lệ) công khai. Tuy nhiên, tôi thấy mã người dùng tạo ra phức tạp hơn và kém mạnh mẽ hơn mã dựa vào việc bắt ngoại lệ: void fill (vector <Date> & aa) e (cin) cin >> d catch (Date: Bad\_date) {"y ror nandiling tiếp tục; aa.push back {d); xem $ 4.4.2 Tuy nhiên, việc kiểm tra xem bộ giá trị a (d, m, y} có phải là ngày hợp lệ không không phải là một phép tính phụ thuộc vào biểu diễn của Ngày, vì vậy tôi đã triển khai is\_valid) theo hàm trợ giúp: bool Date :: is\_valid ( ) return is\_date (d, m, y); Tại sao có cả is\_valid0 và is\_date ()? Trong ví dụ đơn giản này, chúng ta có thể quản lý chỉ với một, nhưng có thể tưởng tượng các hệ thống trong đó là ngày) (như ở đây) kiểm tra xem a (d, m, y> tuple đại diện cho ngày hợp lệ và đâu là\_valid) thực hiện kiểm tra bổ sung xem điều đó ngày tháng có thể được trình bày một cách hợp lý. Ví dụ: is\_valid) có thể từ chối các ngày từ trước khi lịch hiện đại được sử dụng phổ biến. Như thường thấy đối với các loại bê tông đơn giản như vậy, các định nghĩa về các hàm thành viên của Date khác nhau giữa mức nhỏ và không quá phức tạp. Ví dụ: inline int Date :: day) const return d;

Ngày & Ngày: iadd\_month (int n) If (nw = 0) trả về giá trị này; 1 (n> 0) {số nguyên năm int mm = static\_castdnt> (m) + n% 12; số tháng trước Lưu ý W: dec được biểu diễn bằng 12 nt dena\_y z f (12 <mm) + \* deita y; 12; mm. xử lý trường hợpS mà tháng mm không có ngày d. y + delta\_yi m-static cast <Tháng> {mm); trả lại W. xử lý âm n .. return this; Tôi sẽ không gọi mã cho thêm tháng) đẹp. Trên thực tế, nếu tôi thêm tất cả các chi tiết, nó thậm chí có thể đạt đến độ phức tạp của mã tương đối đơn giản trong thế giới thực. 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, nguyên nhân là do biểu diễn d, m, y không thuận tiện cho máy tính như đối với chúng ta. Một đại diện tốt hơn (hoặc cho nhiều mục đích) sẽ chỉ đơn giản là một số sau ngày kể từ "ngày 0 được xác định (ví dụ, tháng 1, 1970). Điều đó sẽ làm cho việc tính toán Ngày trở nên đơn giản với chi phí phức tạp trong kết quả cung cấp phù hợp với con người. Lưu ý rằng việc gán và khởi tạo bản sao được cung cấp theo mặc định ($ 16,2,2). Ngoài ra, Date không cần trình hủy vì Date không sở hữu tài nguyên và không yêu cầu dọn dẹp khi nó vượt ra khỏi phạm vi ($ [3.2.1.2](https://l.facebook.com/l.php?u=http://3.2.1.2/?fbclid=IwAR1721AhOY7MKEk_WxGrH008aoxhq3nQYvQTM4rfFtqKYdnvPUq3PJa9bbE&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank)). 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 Ja.b) hoặc [b, .a) bool is leapyear (int y); bool là ngày tháng (int d, Tháng m, int y); const Ngày & ngày mặc định); Date next\_weekday (Ngày dji Ngày thứ bảy tiếp theo (Ngày d);

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 đối với đại diện. Làm thế nào những chức năng như vậy được "liên kết" với lớp Date? 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 lớp Date. Người dùng cần Ngày sẽ cung cấp tất cả chúng bằng cách bao gồm tệp xác định giao diện (S15.2.2). Ví dụ: #include "Date.h" Ngoài ra (hoặc cách khác), chúng ta có thể làm cho liên kết rõ ràng bằng cách bao bọc lớp và các hàm trợ giúp của nó trong không gian tên ($ 14.3.1) không gian tên Chrono ( W phương tiện để xử lý lớp thời gian Ngày {P int diff (Ngày a, Ngày b); Dooseapyearnt y bool is\_date (int d, Tháng m, int y const Ngày & default \_date (|; Date next\_weekd ay (Ngày d); Ngày thứ bảy tiếp theo (Ngày d); Không gian tên Chrono 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 chúng. 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 helper phải được định nghĩa ở đâu đó: bool Chrono: is \_date (int d, Month m, int y) int ndays; switch (m) case Tháng :: feb: ndays = 28 + 1sleapyear (y break; case Tháng :: apr: case Tháng: jun: case Tháng :: sep: case Tháng :: nov: naays break; case Tháng: / jan : case Tháng :: mar: case Tháng :: may: case Tháng: jul: case Tháng: aug: case Tháng :: oct: case Tháng :: dec: ndays detaur return false; return 1e = d && de = ndays;

m cố tình hơi hoang tưởng ở đây. Một Tháng không nên nằm ngoài phạm vi jan đến dec, nhưng nó có thể xảy ra (ai đó có thể đã cẩu thả với một diễn viên), vì vậy tôi kiểm tra. Ngày default\_ rắc rối cuối cùng trở thành: const Date & Chrono :: default\_date () static Date d (1, Month: jan, 1970); trả lại d; 16.3.3 Người vận hành quá tải Nó thường hữu ích khi thêm các chức năng để kích hoạt ký hiệu thông thường. Ví dụ: toán tử ==) 0 xác định toán tử bình đẳng, ==, hoạt động cho Ngày: W đẳng thức toán tử bool nội dòng = (Ngày a, Ngày b trả về a.day) == b.day) && a.month) = b.month () && ayear (== b.year (); Các ứng cử viên rõ ràng khác là: toán tử bool equaty bool operator> (Date, Date); W lớn hơn Ngày & toán tử ++ (Ngày & d) {return d.add\_ day (1); } Ngày & toán tử - (Ngày & d) {return d.add\_day (-1); ) Ngày tăng lên một ngày giảm Ngày một ngày Ngày & toán tử + = (Ngày & d, int n) {return d.add\_day (n); ) thêm n ngày Ngày & toán tử - = {Ngày & d, int n) {return d.add\_day (-n); } 7 trừ đi n ngày Toán tử ngày (Ngày d, int n) {return d + = n; Ngày khai thác- (Ngày d, int n) {return d + = n, W cộng n adays ngày trừ đi l sản lượng d Tôi đọc thành d ostream & operatoree (ostream &, Date d); lstream & operator >> (istream &, Date & 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). Đối với Date, những toán tử này có thể được coi là những tiện ích đơn thuần. Tuy nhiên, đối với nhiều loại - chẳng hạn như số phức ($ l8,3), vectơ ($ 4,4.) Và các đối tượng giống như hàm (95.4.3, S19.2.2) - việc sử dụng các toán tử thông thường 1 nên rất cố gắng trong tâm trí mọi người rằng định nghĩa của họ gần như là bắt buộc. Quá tải toán tử được thảo luận trong Chương 18. Đối với Date, tôi đã bị cám dỗ để cung cấp + = và - = làm như vậy, tôi sẽ tuân theo một thành ngữ phổ biến ($ [3.2.1.1](https://l.facebook.com/l.php?u=http://3.2.1.1/?fbclid=IwAR3_33iH46Ag_jj6kOHBv9pd3yGVq_yWMRQMgbrMtfIjXvVgr6VQDb8_cc0&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank)). các hàm thành viên thay vì add\_day (). Tôi đã Lưu ý rằng việc gán và khởi tạo sao chép được cung cấp theo mặc định (S16.3, Ș17.3.3)

16.3.4 Tầm quan trọng của Lớp Concrete gọi các kiểu đơn giản do người dùng định nghĩa, chẳng hạn như Ngày, kiểu cụ thể để phân biệt chúng với các lớp trừu tượng ($ 3,2,2) và phân cấp lớp ($ 20,4), và cũng để nhấn mạnh sự tương tự của chúng với buill-in ypes chẳng hạn như int và char. Các lớp bê tông được sử dụng giống như các loại tích hợp sẵn. Các kiểu bê tông còn được gọi là kiểu giá trị và lập trình hướng giá trị sử dụng của chúng. 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 (S3.2.4, Chương 21). Mục đích của loại bê tông là làm tốt và hiệu quả một việc duy nhất, tương đối đơn giản. Thông thường, mục đích không phải là cung cấp cho người dùng các phương tiện để sửa đổi hành vi của một loại cụ thể. Đặc biệt, các loại bê tông không nhằm mục đích hiển thị hành vi đa hình thời gian chạy (xem $ 3.2.3, S20.3.2). 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 "làm lại kiểu cụ thể, bạn sử dụng nó trong việc triển khai kiểu mới của mình chính xác như bạn đã sử dụng kiểu int. Ví dụ: ciass Date and\_ume t private: Ngày d; Thời gian t; pubc và thời gian (Ngày d, Thời gian t) Ngày và giờ (int d, Date :: Tháng m, int y, Time t) and\_timent a, 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 của Vec từ vector ([84.4.1.2](https://l.facebook.com/l.php?u=http://84.4.1.2/?fbclid=IwAR1rBd_N0LDe_OD8nOo49Tep90qPSOuKE52moCcgohWTU7u7nRTDsYRFrlc&h=AT28eZlhXVeNhqRByHi50MkyxdQkd-H4BHob4RhF7KQo41Hm1ItW4dqWsju7lb3fMA9A_UIXz5tQX_rE4n6SjMY-twOkttCc2LtNXd-b6cyNj6qXL89c5AqwSv3RxjfRHmOLuFckj-bi_A3hux98hg" \t "https://www.facebook.com/_blank)) là một ví dụ về điều này. Tuy vậy. dẫn xuất từ ​​một lớp bê tông nên được thực hiện cẩn thận và hiếm khi do thiếu các nút ảo và thông tin loại vôi chạy (S17.5.14, Chapler 2) Với một trình biên dịch hợp lý tốt, một lớp cụ thể như Date không phải chịu chi phí ẩn theo 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\_something (Ngày d) Điều này ít có khả năng bị hiểu nhầm hoặc bị lạm dụng 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

các tập hợp đơn giản của các loại dựng sẵn. Ngoài ra, việc thiếu "loại thông thạo 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 Biểu diễn các khái niệm dưới dạng các lớp; 16,1 đô la. Tách giao diện của một lớp khỏi việc triển khai nó; 16,1 đô la. [2] 13] Chỉ sử dụng dữ liệu công khai (cấu trúc) khi nó thực sự chỉ là dữ liệu và không có bất biến nào có ý nghĩa đối với các thành viên dữ liệu; $ 16,2,4 Định nghĩa một phương thức khởi tạo để xử lý việc khởi tạo các đối tượng; §16.2.5. 41 5 Theo mặc định, khai báo các hàm tạo một đối số rõ ràng; 16,2,6 đô la. Khai báo một hàm thành viên không sửa đổi trạng thái của đối tượng const; 16,2,9 đô la. 6 (7 Loại bê tông là loại đơn giản nhất của lớp. Nếu có thể, thích một kiểu cụ thể hơn các lớp phức tạp hơn và các cấu trúc dữ liệu đơn giản; §16.3. Chỉ biến một hàm thành thành viên nếu nó cần truy cập trực tiếp vào biểu diễn của một lớp; 8 $ 16,3,2. 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 đô la. [10] Biến một hàm thành viên không sửa đổi giá trị của đối tượng của nó thành một hàm thành viên const; 16,2,9,1 đô la. [11] Tạo một hàm cần truy cập vào biểu diễn của một lớp nhưng không cần được gọi cho một đối tượng cụ thể thành một hàm thành viên tĩnh; 16,2,12 đô la. 19