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

**18**



***Khi nào tôi sử dụng một từ nó chỉ có nghĩa là những***

***gì tôi chọn nó có nghĩa là - không hơn không kém.***

***- Humpty Dumpty***

* **Giới thiệu**
* **Chức năng của nhà điều hành**

**Toán tử nhị phân và đơn nguyên; Ý nghĩa được xác định trước cho các nhà khai thác; Các toán tử và các loại do người dùng xác định; Đối tượng Chuyền; Toán tử trong Không gian tên**

**• Một loại số phức**

**Các nhà điều hành thành viên và không phải thành viên; Số học Chế độ Hỗn hợp; Chuyển đổi; Chữ viết; Chức năng của Accessor; Chức năng của người trợ giúp**

**• Chuyển đổi loại**



**Nhà điều hành chuyển đổi**

**• Lời khuyên**

**18.1 Giới thiệu**

**Mọi lĩnh vực kỹ thuật - và hầu hết các lĩnh vực phi kỹ thuật - đều phát triển các ký hiệu viết tắt thông thường để thuận tiện cho việc trình bày và thảo luận liên quan đến các khái niệm được sử dụng thường xuyên. Ví dụ, vì quen biết lâu,**

**x + y∗z**

**rõ ràng hơn đối với chúng tôi**

**nhân y với z và cộng kết quả với x**

**Thật khó để đánh giá quá cao tầm quan trọng của ký hiệu súc tích đối với các phép toán thông thường.**

**Giống như hầu hết các ngôn ngữ khác, C ++ hỗ trợ một tập hợp các toán tử cho các kiểu tích hợp của nó. Tuy nhiên, hầu hết các khái niệm mà các toán tử được sử dụng thông thường không phải là các kiểu tích hợp sẵn trong C ++, vì vậy chúng phải**



**528** **Người vận hành quá tải** **Chương 18**

**được biểu diễn dưới dạng các kiểu do người dùng xác định. Ví dụ: nếu bạn cần số học phức tạp, đại số ma trận, tín hiệu logic hoặc chuỗi ký tự trong C ++, bạn sử dụng các lớp để biểu diễn các khái niệm này. Việc xác định các toán tử cho các lớp như vậy đôi khi cho phép lập trình viên cung cấp một ký hiệu thông thường và thuận tiện hơn để thao tác các đối tượng hơn là có thể đạt được chỉ bằng cách sử dụng ký hiệu chức năng cơ bản. Xem xét:**

**lớp phức tạp {**

**gấp đôi lại, im;**

**// *rất đơn giản hóa phức tạp***

**công cộng:**

**complex (double r, double i): re {r}, im {i} {}**

**complex operator + (complex);**

**toán tử phức tạp∗(NS**

**};**

**Điều này xác định một cách triển khai đơn giản của khái niệm số phức. MỘTphức tạp được biểu thị bằng một cặp số dấu phẩy động có độ chính xác kép được thao tác bởi các toán tử + và ∗. Lập trình viên định nghĩaphức tạp :: operator + () và phức tạp :: toán tử∗() để cung cấp ý nghĩa cho +và ∗, tương ứng. Ví dụ, nếuNS và NS thuộc loại phức tạp, b + c có nghĩa b. người điều hành + (c). Bây giờ chúng ta có thể gần đúng với cách giải thích thông thường củaphức tạp biểu thức:**

**void f ()**

**{**

**complex a = complex {1,3.1};**

**phức b {1,2, 2};**

**phức c {b};**

**a = b + c;**

**b = b + c∗Một;**

**c = a∗b + phức (1,2);**

**}**



**Các quy tắc ưu tiên thông thường**

**Lưu ý rằng gam C ++**

**ở phía bên phải của bài tập:**

**void g (phức a, phức b) {**

**a = {1,2};** **// *OK: phía bên phải của nhiệm vụ*//**

**a + = {1,2};** ***OK: phía bên phải của nhiệm vụ*// *lỗi***

**b = a + {1,2};** ***cú pháp***

**b = a + phức {1,2}; //*VÂNG*g**

**(a, {1,2});** **// *OK: một đối số hàm được coi là bộ khởi tạo*// *lỗi cú***

**{a, b} = {b, a};** ***pháp***

**}**

**Dường như không có lý do cơ bản nào để không sử dụng {} ở nhiều nơi hơn, nhưng các vấn đề kỹ thuật khi viết ngữ pháp cho phép {} ở mọi nơi trong một biểu thức (ví dụ: làm thế nào bạn biết nếu một { sau dấu chấm phẩy là phần bắt đầu của một biểu thức hay một khối?) và cũng đưa ra các thông báo lỗi tốt dẫn đến việc sử dụng hạn chế hơn {} trong các biểu thức.**



**Mục 18.1** **Giới thiệu** **529**

**Nhiều ứng dụng rõ ràng nhất của việc nạp chồng toán tử là đối với các kiểu số. Tuy nhiên, tính hữu ích của các toán tử do người dùng định nghĩa không bị giới hạn ở các kiểu số. Ví dụ, việc thiết kế các giao diện chung và trừu tượng thường dẫn đến việc sử dụng các toán tử như->, [], và ().**

**18.2 Các chức năng của nhà điều hành**

**Các hàm xác định ý nghĩa cho các toán tử sau (§10.3) có thể được khai báo:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **+** | **-** | **∗** | **/** | **%** | **ˆ** | **&** |
| **|** | **˜** | **!** | **=** | **<** | **>** | **+ =** |
| **- =** | **∗=** | **/ =** |  |  |  |  |
| **<<** | **>>** | **>> =** |  |  |  |  |
| **> =** | **&&** | **||** | **+ +** | **- -** | **-> ∗** | **,** |
| **->** | **[]** | **()** | **Mới** | **Mới[]** | **xóa bỏ** | **xóa bỏ[]** |

**Người dùng không thể xác định các toán tử sau:**

* **giải quyết phạm vi (§6.3.4, §16.2.12) lựa**

**.chọn thành viên (§8.2)**

**.∗** **lựa chọn thành viên thông qua con trỏ đến thành viên (§20.6)**

**Chúng lấy một cái tên, thay vì một giá trị, làm toán hạng thứ hai của chúng và cung cấp phương tiện chính để đề cập đến các thành viên. Cho phép chúng bị quá tải sẽ dẫn đến sự kém tinh tế [Stroustrup, 1994]. Không thể quá tải '' toán tử '' được đặt tên vì chúng báo cáo sự kiện cơ bản về toán hạng của chúng:**

**kích thước** **kích thước của đối tượng (§6.2.8) căn**

**alignof** **chỉnh của đối tượng (§6.2.9)type\_info**

**người đánh chữ** **của một đối tượng (§22.5)**

**Cuối cùng, toán tử biểu thức điều kiện bậc ba không thể được nạp chồng (không vì lý do đặc biệt cơ bản):**



**?:** **ev có điều kiện**

**Ngoài ra, loại chuyển đổi phụ cú pháp**

**theo cú pháp sáng do người dùng xác**

**định thành một loại NS (§18.4).**

**Không thể xác định mã thông báo toán tử mới, nhưng bạn có thể sử dụng ký hiệu lệnh gọi hàm khi tập hợp toán tử này không đủ. Ví dụ, sử dụngpow (), không phải ∗∗. Những hạn chế này có vẻ hà khắc, nhưng các quy tắc linh hoạt hơn có thể dễ dẫn đến sự mơ hồ. Ví dụ, xác định một toán tử∗∗ nghĩa là lũy thừa có vẻ là một nhiệm vụ hiển nhiên và dễ dàng, nhưng hãy nghĩ lại. Nên∗∗ ràng buộc bên trái (như ở Fortran) hay bên phải (như ở Algol)? Nên biểu thứcMột∗∗P được hiểu làMột∗(∗P) hoặc như (Một)∗∗(P)? Có giải pháp cho tất cả các câu hỏi kỹ thuật như vậy. Tuy nhiên, điều không chắc chắn nhất là nếu áp dụng các quy tắc kỹ thuật tinh vi sẽ dẫn đến mã dễ đọc và dễ bảo trì hơn. Nếu nghi ngờ, hãy sử dụng một hàm được đặt tên.**

**Tên của một hàm toán tử là từ khóa nhà điều hành theo sau là chính toán tử, ví dụ, toán tử <<. Một hàm toán tử được khai báo và có thể được gọi như bất kỳ hàm nào khác. Việc sử dụng toán tử chỉ là cách viết tắt cho một lệnh gọi rõ ràng của hàm toán tử. Ví dụ:**



**530** **Người vận hành quá tải** **Chương 18**

**void f (phức a, phức b) {**

**phức c = a + b; phức d =** **// *tốc ký***

**a.operator + (b);** **// *cuộc gọi rõ ràng***

**}**

**Đưa ra định nghĩa trước đây của phức tạp, hai bộ khởi tạo là đồng nghĩa.**

**18.2.1 Toán tử nhị phân và đơn nguyên**

**Một toán tử nhị phân có thể được xác định bởi mộttĩnh hàm thành viên lấy một đối số hoặc một hàm khác lấy**

**như một trong hai aa.operator @ (bb)**

**xác định cách diễn giải, nếu có, được sử dụng. Ví dụ:**

**lớp X {**

**công cộng:**

**void operator + (int);**

**X (int);**

**};**

**void operator + (X, X);**

**void operator + (X, double);**

**void f (X a)**

**{**

**a + 1;** **// *a. người điều hành + (1)***

**1 + a;** **// *::toán tử + (X (1), a)***

**a + 1,0;** **// *::toán tử + (a, 1,0)***

**}**



**Toán tử một ngôi, cho dù không lấy đối số hay không@, @aa có thể được hiểu là aa.operator @ ()hoặc toán tử @ (aa). Nếu cả hai đều được xác định, giải pháp quá tải (§12.3) sẽ xác định cách giải thích nào, nếu có, được sử dụng. Đối với bất kỳ toán tử đơn vị postfix nào@, aa @ có thể được hiểu là aa.operator @ (int)hoặc operator @ (aa, int). Điều này được giải thích thêm trong §19.2.4. Nếu cả hai đều được xác định, giải pháp quá tải (§12.3) sẽ xác định cách giải thích nào, nếu có, được sử dụng. Một toán tử chỉ có thể được khai báo cho cú pháp được định nghĩa cho nó trong ngữ pháp (§iso.A). Ví dụ: người dùng không thể xác định một% hoặc một con ba ba +. Xem xét:**

**lớp X {**

**công cộng:** **// *các thành viên (với con trỏ này ngầm định):***

**NS∗ nhà điều hành&();**

**Toán tử X & (X);**

**Toán tử X ++ (int);**

**Toán tử X & (X, X);**

**Toán tử X / ();**

**// *tiền tố một ngôi & (địa chỉ của)*//**

***nhị phân & (và)*// *gia số hậu tố (xem***

***§19.2.4)*// *error: ternary***

* ***lỗi: một ngôi /***

**};**



**Mục 18.2.1** **Toán tử nhị phân và đơn nguyên** **531**

* ***các hàm nonmember:***

**Toán tử X− (X);** **// *tiền tố một ngôi trừ***

**Toán tử X− (X, X);** **// *nhị phân trừ***

**Toán tử X −− (X &, int);** **// *giảm hậu tố***

**Toán tử X− ();** **// *lỗi: không có toán***

**Toán tử X− (X, X, X);** ***hạng*// *error: ternary***

**Toán tử X% (X);** **// *error: một%***

**Nhà điều hành [] được mô tả trong §19.2.1, toán tử () trong §19.2.2, toán tử -> trong §19.2.3, toán tử ++ và**

* **- trong §19.2.4, và các toán tử phân bổ và phân bổ trong §11.2.4 và §19.2.5. Các nhà**

**khai thácnhà điều hành**

**(§19.2.3) phải khôngtĩnh**

* **nghĩa mặc định của &&, ||, và , (dấu phẩy) liên quan đến trình tự: toán hạng đầu tiên được đánh giá trước toán hạng thứ hai (và cho && và || toán hạng thứ hai không phải lúc nào cũng được đánh giá). Quy tắc đặc biệt này không áp dụng cho các phiên bản do người dùng xác định&&, ||, và , (dấu phẩy); thay vào đó các toán tử này được xử lý chính xác như các toán tử nhị phân khác.**

**18.2.2 Ý nghĩa được xác định trước cho người vận hành**

* **nghĩa của một số toán tử dựng sẵn được định nghĩa tương đương với một số kết hợp của các toán tử khác trên cùng các đối số. Ví dụ, nếuMột là một int, ++Một có nghĩa a + = 1, điều đó có nghĩa làa = a + 1. Các quan hệ như vậy không giữ cho các toán tử do người dùng xác định trừ khi người dùng định nghĩa chúng. Ví dụ: một trình biên dịch sẽ không tạo ra một định nghĩa vềZ :: toán tử + = () từ các định nghĩa của Z :: opera-**

**tor + () và Z :: toán tử = ().**

**Các nhà khai thác = (phân công), & (address-of), và , (giải trình tự; §10.3.2) có ý nghĩa được xác định trước khi áp dụng cho các đối tượng lớp. Những ý nghĩa được xác định trước này có thể bị loại bỏ ('' bị xóa ''; §17.6.4):**

**lớp X {**



**công cộng:**

* ***...***

**void operator = (const X &) =**

**delete; toán tử void & () = xóa;**

**toán tử void, (const X &) = xóa; //*...***

**};**

**void f (X a, X b) {**

**a = b;** **// *lỗi: không có toán tử = ()*//**

**&Một;** ***lỗi: không có toán tử & ()*//**

**a, b;** ***lỗi: không có toán tử, ()***

**}**

**Ngoài ra, chúng có thể được đưa ra các nghĩa mới bằng các định nghĩa phù hợp.**



**532** **Người vận hành quá tải** **Chương 18**

**18.2.3 Các toán tử và các loại do người dùng xác định**

**Một hàm toán tử phải là một thành viên hoặc có ít nhất một đối số của kiểu do người dùng xác định (các hàm xác định lại Mới và xóa bỏ toán tử không cần). Quy tắc này đảm bảo rằng người dùng không thể thay đổi ý nghĩa của một biểu thức trừ khi biểu thức chứa một đối tượng thuộc loại do người dùng xác định.**

**Đặc biệt, không thể định nghĩa một hàm toán tử hoạt động độc quyền trên con trỏ. Điều này đảm bảo rằng C ++ có thể mở rộng nhưng không thể thay đổi (ngoại trừ các toán tử=, &, và , cho các đối tượng lớp).**

**Một hàm toán tử nhằm chấp nhận một kiểu dựng sẵn (§6.2.1) vì toán hạng đầu tiên của nó không thể là một hàm thành viên. Ví dụ: hãy xem xét thêm một biến phức tạpaa đến số nguyên 2: aa + 2 có thể, với một tuyên bố phù hợp với tôi**

**không có lớp học NS cho bất kỳ**

**các chức năng thành viên sẽ cần thiết để đối phó với 2 + aa và aa + 2. Vì trình biên dịch không biết ý nghĩa của từ do người dùng định nghĩa+, nó không thể giả định rằng toán tử là giao hoán và do đó, diễn giải 2 + aa như aa + 2. Ví dụ này được xử lý thông thường bằng cách sử dụng một hoặc nhiều hàm không phải là phần tử (§18.3.2, §19.4).**

**Các kiểu liệt kê là các kiểu do người dùng định nghĩa để chúng ta có thể xác định các toán tử cho chúng. Ví dụ:**

**enum Day {sun, mon, tue, wed, thu, fri, sat};**

**Day & operator ++ (Day & d) {**

**trả về d = (sat == d)? sun: static\_cast <Ngày> (d + 1);**

**}**

**Mọi biểu thức đều được kiểm tra xem có mơ hồ không. Trong trường hợp toán tử do người dùng định nghĩa cung cấp một cách diễn giải khả thi, biểu thức được kiểm tra theo các quy tắc giải quyết quá tải trong §12.3.**

**18.2.4 Chuyển đối tượng**



**Khi chúng ta xác định một operato**

**a = b + c. Do đó, chúng ta h**

**và cách nó trả về giá trị của nó. Ví dụ: chúng tôi không thể yêu cầu đối số con trỏ và mong đợi người lập trình sử dụng toán tử address-of hoặc trả về một con trỏ và mong đợi người dùng bỏ qua nó:∗a = & b + & c không thể chấp nhận được.**

**Đối với các đối số, chúng ta có hai lựa chọn chính (§12.2):**

* **Giá trị chuyển tiếp**
* **Tham khảo qua**

**Đối với các đối tượng nhỏ, chẳng hạn như từ một đến bốn từ, gọi theo giá trị thường là một phương án thay thế khả thi và thường là phương án mang lại hiệu suất tốt nhất. Tuy nhiên, hiệu suất của việc truyền và sử dụng đối số phụ thuộc vào kiến trúc máy, quy ước giao diện trình biên dịch (Giao diện nhị phân ứng dụng; ABI) và số lần một đối số được truy cập (hầu như luôn luôn nhanh hơn để truy cập một đối số được truyền bằng giá trị hơn một đối số được truyền qua thẩm quyền giải quyết). Ví dụ, giả sử rằng mộtChỉ trỏ được đại diện như một cặp NSNS:**

**void Point :: operator + = (Point delta);** **// *truyền qua giá trị***



**Mục 18.2.4** **Đi qua các đối tượng** **533**

**Các đối tượng lớn hơn, chúng tôi chuyển qua tham chiếu. Ví dụ, bởi vìMa trận (một ma trận đơn giản của képNS; §17.5.1) rất có thể lớn hơn một vài từ, chúng tôi sử dụng tham chiếu chuyển qua:**

**Toán tử ma trận + (const Matrix &, const Matrix &);** **// *pass-by-const-reference***

**Đặc biệt, chúng tôi sử dụng hăng sô các tham chiếu để truyền các đối tượng lớn không có nghĩa là được sửa đổi bởi hàm được gọi (§12.2.1).**

**Thông thường, một toán tử trả về một kết quả. Trả lại một con trỏ hoặc một tham chiếu đến một đối tượng mới được tạo thường là một ý tưởng rất tồi: việc sử dụng một con trỏ gây ra các vấn đề về ký hiệu và tham chiếu đến một đối tượng trên kho lưu trữ miễn phí (cho dù bằng con trỏ hay bằng tham chiếu) dẫn đến các vấn đề quản lý bộ nhớ. Thay vào đó, hãy trả về các đối tượng theo giá trị. Đối với các đối tượng lớn, chẳng hạn nhưMa trận, xác định các hoạt động di chuyển để thực hiện chuyển giao như vậy**

**Toán tử ma trận + (const M {**

**Ma trận res {a};**

**trả về res + = b;**

**}**

**Lưu ý rằng các toán tử trả về một trong các đối tượng của chúng có thể - và thường là - trả về một tham chiếu. Ví dụ, chúng tôi có thể xác địnhMa trậnnhà điều hành của + = như thế này:**

**Ma trận & Ma trận :: operator + = (const Matrix & a) {** **// *trả về từng tham chiếu***

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

**ném std :: exception ("xấu Ma trận + = đối số");**

**kép∗ p = độ cao; kép∗ q = a.elem;**

**kép∗ end = p + dim [0]∗mờ [1];**

**trong khi (p! = end)**

**∗p ++ + = ∗q ++**



**trở lại ∗cái này;**

**}**

**Điều này đặc biệt phổ biến đối với các hàm toán tử được triển khai với tư cách là thành viên.**

**Nếu một hàm chỉ đơn giản là chuyển một đối tượng cho một hàm khác, thì nên sử dụng đối số tham chiếu rvalue (§17.4.3, §23.5.2.1, §28.6.3).**

**18.2.5 Các toán tử trong Không gian tên**

**Toán tử hoặc là một thành viên của một lớp hoặc được định nghĩa trong một số không gian tên (có thể là không gian tên chung). Hãy xem xét phiên bản đơn giản này của chuỗi I / O từ thư viện tiêu chuẩn:**

**không gian tên std {** **// *std đơn giản hóa***

**chuỗi lớp {**

* ***...***

**};**



|  |  |  |
| --- | --- | --- |
| **534 Người vận hành quá tải** | **Chương 18** |  |
| **lớp học ostream {** |  |  |
| **// *...*** | **// *xuất chuỗi kiểu C*** |  |
| **ostream & operator << (const char∗);** |  |
| **};** |  |  |
| **cout ostream extern;** |  |  |
| **ostream & operator << (ostream &, const string &); } //** | **// *đầu ra std :: string*** |  |
| ***không gian tên std*** |  |  |
| **int main ()** |  |  |
| **{** |  |  |
| **const char∗ p = "Hel std ::** |  |  |
| **string s =" world ";** |  |  |
| **std :: cout << p << "," << s << "! \ n";** |  |  |
| **}** |  |  |

**Đương nhiên, điều này viết ra Chào thế giới!. Nhưng tại sao? Lưu ý rằng tôi không làm mọi thứ từstdcó thể truy cập bằng cách viết:**

**sử dụng không gian tên std;**

**Thay vào đó, tôi đã sử dụng std :: tiền tố cho dây và cout. Nói cách khác, tôi đã thực hiện hành vi tốt nhất của mình và không làm ô nhiễm không gian tên chung hoặc theo cách khác, đưa ra các phụ thuộc không cần thiết.**

**Toán tử đầu ra cho chuỗi kiểu C là một thành viên của std :: ostream, vì vậy theo định nghĩa**

**std :: cout << p**

**có nghĩa**

**std :: cout.operator << (p)**

**Tuy vậy, std :: ostream không có chức năng thành viên để xuất ra một std :: string, vì thế**



**std :: cout << s**

**có nghĩa**

**toán tử << (std :: cout, s)**

**Các toán tử được định nghĩa trong không gian tên có thể được tìm thấy dựa trên kiểu toán hạng của chúng cũng như các hàm có thể được tìm thấy dựa trên kiểu đối số của chúng (§14.2.4). Đặc biệt,cout nằm trong không gian tên std, vì thế std được cân nhắc khi tìm kiếm một định nghĩa phù hợp về <<. Theo cách đó, trình biên dịch tìm và sử dụng:**

**std :: operator << (std :: ostream &, const std :: string &)**

**Hãy xem xét một toán tử nhị phân @. Nếu nhưNS thuộc loại NS và y thuộc loại Y, x @ y được giải quyết như thế này:**

* **Nếu như NS là một lớp học, hãy tìm nhà điều hành@ là một thành viên của NS hoặc là thành viên của một cơ sở của NS; và**
* **tìm kiếm các khai báo của nhà điều hành@ trong bối cảnh xung quanh x @ y; và**
* **nếu như NS được xác định trong không gian tên n, tìm kiếm khai báo của nhà điều hành@ trong n; và**
* **nếu như Y được xác định trong không gian tên NS, tìm kiếm khai báo của nhà điều hành@ trong NS.**

**Tuyên bố cho một số nhà điều hành@s có thể được tìm thấy và các quy tắc giải quyết quá tải (§12.3) được sử dụng để tìm kết quả phù hợp nhất, nếu có. Cơ chế tra cứu này chỉ được áp dụng nếu nhà điều hành có ít nhất một**



**Mục 18.2.5** **Toán tử trong Không gian tên** **535**

**toán hạng của một kiểu do người dùng xác định. Do đó, các chuyển đổi do người dùng xác định (§18.3.2, §18.4) sẽ được xem xét.**

**Lưu ý rằng bí danh kiểu chỉ là một từ đồng nghĩa và không phải là một kiểu riêng biệt do người dùng xác định (§6.5).**

**Các toán tử đơn phân được giải quyết một cách tương tự.**

**Lưu ý rằng trong tra cứu nhà điều hành không có ưu tiên nào dành cho các thành viên hơn là những người không phải là thành viên. Điều này khác với tra cứu các hàm được đặt tên (§14.2.4). Việc thiếu ẩn các toán tử đảm bảo rằng các toán tử tích hợp không bao giờ không thể truy cập được và người dùng có thể cung cấp các ý nghĩa mới cho một toán tử mà không cần sửa đổi các khai báo lớp hiện có. Ví dụ:**

**Toán tử X! (X);**

**cấu trúc Z {**

**Toán tử Z! ();**

**X f (X x) {/ \* *...* \* / trả về int f (int x)**

**{/ \* *...* \* / quay lại! x; }** **// *gọi tích hợp sẵn! cho ints***

**};**

**Đặc biệt, tiêu chuẩn iostream thư viện xác định << các chức năng thành viên để xuất ra các kiểu tích hợp và người dùng có thể xác định << để xuất ra các kiểu do người dùng xác định mà không cần sửa đổi lớp ostream (§38.4.2).**

**18.3 Một loại số phức**

**Việc triển khai các số phức được trình bày trong §18.1 là quá hạn chế để làm hài lòng bất kỳ ai. Ví dụ:**

**chúng tôi mong đợi điều này hoạt động:**

**void f ()**

**{**

**phức a {1,2};**

**phức b {3};**

**phức c {a + 2.3};**

**phức d {2 + b};**



**b = c∗2∗NS;**

**}**

**Ngoài ra, chúng tôi mong muốn được cung cấp thêm một số toán tử khác, chẳng hạn như == để so sánh và**

* **cho đầu ra và một tập hợp các hàm toán học phù hợp, chẳng hạn như tội() và sqrt (). Lớpphức tạp là một loại bê tông, vì vậy thiết kế của nó tuân theo các hướng dẫn từ §16.3. Ngoài ra, những người sử dụng**

**số học phức tạp phụ thuộc rất nhiều vào các toán tử nên định nghĩa củaphức tạp phát huy hầu hết các quy tắc cơ bản để nạp chồng toán tử.**

**Các phức tạp loại được phát triển trong phần này sử dụng kép cho các đại lượng vô hướng của nó và gần tương đương với thư viện tiêu chuẩn phức tạp <double> (§40.4).**

**18.3.1 Các nhà điều hành thành viên và không phải thành viên**

**Tôi muốn giảm thiểu số lượng hàm thao tác trực tiếp với biểu diễn của một đối tượng. Điều này có thể đạt được bằng cách chỉ xác định các toán tử vốn đã sửa đổi giá trị của đối số đầu tiên của chúng, chẳng hạn như+ =, trong chính lớp học. Các toán tử chỉ tạo ra một giá trị mới dựa trên các giá trị của các đối số của chúng, chẳng hạn như+, sau đó được định nghĩa bên ngoài lớp và sử dụng các toán tử thiết yếu trong việc triển khai chúng:**



**536** **Người vận hành quá tải** **Chương 18**

**lớp phức tạp {**

**gấp đôi lại, im;**

**công cộng:**

**phức & toán tử + = (phức a); //*...*** **// *cần quyền truy cập để đại diện***

**};**

**toán tử phức + (phức a, phức b) {**

**trả về a + = b; //*truy cập biểu diễn thông qua + =***

**}**

**Các đối số cho điều này opera**

**Với những khai báo này, chúng ta có thể viết:**

**void f (phức x, phức y, phức z) {**

**phức r1 {x + y + z}; //*r1 = operator + (operator + (x, y), z)***

|  |  |  |
| --- | --- | --- |
| **phức r2 {x};** | | **// *r2 = x*** |
| **r2** | **+ = y;** | **// *r2.operator + = (y)*** |
| **r2** | **+ = z;** | **// *r2.operator + = (z)*** |

**}**

**Ngoại trừ sự khác biệt về hiệu quả có thể xảy ra, các tính toán của r1 và r2 là tương đương.**

**Các toán tử gán tổng hợp chẳng hạn như + = và ∗= có xu hướng định nghĩa đơn giản hơn so với các đối tác '' đơn giản '' của chúng + và ∗. Điều này làm cho hầu hết mọi người ngạc nhiên lúc đầu, nhưng nó xuất phát từ thực tế là ba đối tượng có liên quan đến một+ hoạt động (hai toán hạng và kết quả), trong khi chỉ có hai đối tượng tham gia vào một + = hoạt động. Trong trường hợp thứ hai, hiệu quả thời gian chạy được cải thiện bằng cách loại bỏ nhu cầu về các biến tạm thời. Ví dụ:**



**nội tuyến phức tạp & phức tạp {**

**re + = a.re;**

**im + = a.im;**

**trở lại ∗cái này;**

**}**

**Điều này không yêu cầu một biến tạm thời để giữ kết quả của phép cộng và đơn giản để trình biên dịch có thể nội dòng một cách hoàn hảo.**

**Một trình tối ưu hóa tốt sẽ tạo ra mã gần với mã tối ưu để sử dụng + nhà điều hành cũng. Tuy nhiên, không phải lúc nào chúng ta cũng có một trình tối ưu hóa tốt và không phải tất cả các loại đều đơn giản nhưphức tạp, vì vậy §19.4 thảo luận về các cách xác định các toán tử có quyền truy cập trực tiếp vào biểu diễn của các lớp.**

**18.3.2 Số học chế độ hỗn hợp**

**Để đối phó với 2 + z, ở đâu z là một phức tạp, chúng ta cần xác định toán tử + để chấp nhận các toán hạng của các kiểu khác nhau. Theo thuật ngữ Fortran, chúng ta cần*số học chế độ hỗn hợp*. Chúng tôi có thể đạt được điều đó đơn giản bằng cách thêm các phiên bản thích hợp của các toán tử:**



**Mục 18.3.2** **Số học chế độ hỗn hợp** **537**

**lớp phức tạp {**

**gấp đôi lại, im;**

**công cộng:**

**phức & toán tử + = (phức a) {**

**re + = a.re;**

**im + = a.im;**

**trở lại ∗cái này;**

**}**

**phức tạp & toán tử +**

**{**

**re + = a;**

**trở lại ∗cái này;**

**}**

* ***...***

**};**

**Ba biến thể của toán tử + () có thể được xác định bên ngoài phức tạp:**

**toán tử phức + (phức a, phức b) {**

**trả về a + = b; //*cuộc gọi phức tạp :: operator + = (complex)***

**}**

**toán tử phức + (phức a, kép b) {**

**return {a.real () + b, a.imag ()};**

**}**



**toán tử phức tạp + (awl {**

**return {a + b.real (), b.imag ()};**

**}**

**Các chức năng truy cập thực() và tưởng tượng () được định nghĩa trong §18.3.6.**

**Với những tuyên bố này về +, chúng tôi có thể viết:**

**void f (phức x, phức y) {**

**auto r1 = x + y;**

**auto r2 = x + 2;**

**auto r3 = 2 + x;**

**auto r4 = 2 + 3;**

**// *gọi nhà điều hành + (phức tạp, phức tạp)*//**

***gọi nhà điều hành + (phức tạp, kép)*// *gọi***

***nhà điều hành + (kép, phức tạp)*// *phép cộng***

***số nguyên cài sẵn***

**}**

**Tôi đã thêm phép cộng số nguyên cho đầy đủ.**



**538** **Người vận hành quá tải** **Chương 18**

**18.3.3 Chuyển đổi**

**Để đối phó với các nhiệm vụ và khởi tạo phức tạp các biến có đại lượng vô hướng, chúng ta cần chuyển đổi một đại lượng vô hướng (số nguyên hoặc số dấu phẩy động) thành phức tạp. Ví dụ:**

**phức b {3}; //*nghĩa là b.re = 3, b.im = 0***

**void comp (phức x)**

**{**

**x = 4;** **// *nghĩa là x.re = 4, x.im = 0***

* ***...***

**}**

**Chúng ta có thể đạt được điều đó bằng cách**

**đối số đơn chỉ định một chuyển đổi từ kiểu đối số của nó sang kiểu của phương thức khởi tạo. Ví dụ:**

**lớp phức tạp {**

**gấp đôi lại, im;**

**công cộng:**

**complex (double r): re {r}, im {0} {} // *xây dựng một khu phức hợp từ một đôi*// *...***

**};**

**Hàm tạo chỉ định cách nhúng truyền thống của đường thực trong mặt phẳng phức.**

**Một hàm tạo là một đơn thuốc để tạo một giá trị của một kiểu nhất định. Hàm khởi tạo được sử dụng khi một giá trị của một kiểu được mong đợi và khi một giá trị đó có thể được tạo bởi một hàm tạo từ giá trị được cung cấp dưới dạng bộ khởi tạo hoặc giá trị được gán. Do đó, một hàm tạo yêu cầu một đối số duy nhất không cần được gọi một cách rõ ràng. Ví dụ:**

**phức b {3};**



**có nghĩa**

**phức b {3,0};**

**Chuyển đổi do người dùng xác định chỉ được áp dụng hoàn toàn nếu nó là duy nhất (§12.3). Nếu bạn không muốn một hàm tạo được sử dụng ngầm, hãy khai báo nórõ ràng (§16.2.6).**

**Đương nhiên, chúng ta vẫn cần hàm tạo có hai képs và một hàm khởi tạo mặc định-**

**izing a phức tạp đến {0,0} cũng hữu ích:**

**lớp phức tạp {**

**gấp đôi lại, im;**

**công cộng:**

**complex (): re {0}, im {0} {} complex (double r):**

**re {r}, im {0} {} complex (double r, double i): re**

**{r}, im {i} {} // *...***

**};**

**Sử dụng các đối số mặc định, chúng ta có thể viết tắt:**



**Mục 18.3.3** **Chuyển đổi** **539**

**lớp phức tạp {**

**gấp đôi lại, im;**

**công cộng:**

**complex (double r = 0, double i = 0): re {r}, im {i} {} //**

***...***

**};**

**Theo mặc định, sao chép phức tạp giá trị được định nghĩa là sao chép phần thực và phần ảo (§16.2.2).**

**Ví dụ:**

**void f ()**

**{**

**phức tạp z;**

**phức x {1,2};**

**phức y {x}; //*y cũng có giá trị {1,2}*z = x; //*z***

***cũng có giá trị {1,2}***

**}**

**18.3.3.1 Chuyển đổi toán hạng**

**Chúng tôi đã xác định ba phiên bản của mỗi trong bốn toán tử số học tiêu chuẩn:**

**toán tử phức tạp + (phức tạp, phức**

**tạp); toán tử phức tạp + (phức tạp,**

**kép); toán tử phức + (kép, phức); //*...***

**Điều này có thể trở nên tẻ nhạt và những gì tẻ nhạt dễ trở thành lỗi. Điều gì sẽ xảy ra nếu chúng ta có ba lựa chọn thay thế cho loại của mỗi đối số cho mỗi hàm? Chúng ta sẽ cần ba phiên bản của mỗi hàm đối số đơn, chín phiên bản của mỗi hàm hai đối số, 27 phiên bản của mỗi hàm đối số, v.v. Thường thì những biến thể này rất giống nhau. Trên thực tế, hầu hết tất cả các biến thể liên quan đến một chuyển đổi đơn giản của argum**



**Sự thay thế cho cung cấp**

**là dựa vào chuyển đổi. Ví dụ, của chúng tôiphức tạp lớp cung cấp một phương thức khởi tạo để chuyển đổi mộtkép đến một phức tạp. Do đó, chúng ta chỉ có thể khai báo một phiên bản của toán tử bình đẳng cho phức tạp:**

**toán tử bool == (phức tạp, phức tạp);**

**void f (phức x, phức y) {**

**x == y;**

**x == 3;**

**3 == y;**

**// *có nghĩa là toán tử == (x, y)*// *có nghĩa***

***là toán tử == (x, complex (3))*// *có nghĩa***

***là toán tử == (phức (3), y)***

**}**

**Có thể có lý do để thích xác định các chức năng riêng biệt. Ví dụ: trong một số trường hợp, chuyển đổi có thể áp đặt chi phí và trong các trường hợp khác, một thuật toán đơn giản hơn có thể được sử dụng cho các loại đối số cụ thể. Trong trường hợp các vấn đề như vậy không đáng kể, việc dựa vào chuyển đổi và chỉ cung cấp biến thể chung nhất của một hàm - cộng với có thể là một vài biến thể quan trọng - chứa**



**phức tạp kiểu.**

**540** **Người vận hành quá tải** **Chương 18**

**sự bùng nổ tổ hợp của các biến thể có thể phát sinh từ số học ở chế độ hỗn hợp.**

**Khi một số biến thể của một hàm hoặc một toán tử tồn tại, trình biên dịch phải chọn biến thể '' đúng '' dựa trên các loại đối số và các chuyển đổi có sẵn (tiêu chuẩn và do người dùng xác định). Trừ khi một kết quả phù hợp nhất tồn tại, một biểu thức không rõ ràng và là một lỗi (xem §12.3).**

**Một đối tượng được xây dựng bằng cách sử dụng hàm tạo trong một biểu thức một cách rõ ràng hoặc ngầm định là tự động và sẽ bị hủy ở cơ hội đầu tiên (xem §10.3.4).**

**Không có chuyển đổi ngầm định nào do người dùng xác định được áp dụng cho phía bên trái của . (hoặc một ->). Đây là**

**trường hợp ngay cả khi . là ẩn ý. Ví dụ:**

**void g (phức z)**

**{**

**3 + z;**

**3. điều hành + = (z);**

**3 + = z;** **// *error: 3 không phải là một đối tượng lớp***

**}**

**Vì vậy, bạn có thể gần đúng khái niệm rằng một toán tử yêu cầu một giá trị làm toán hạng bên trái của nó bằng cách đặt toán tử đó thành một thành viên. Tuy nhiên, đó chỉ là ước tính vì có thể truy cập tạm thời bằng thao tác sửa đổi, chẳng hạn nhưtoán tử + = ():**

**phức x {4,5}**

**phức z {sqrt (x) + = {1,2}};** **// *như tmp = sqrt (x), tmp + = {1,2}***

**Nếu không muốn chuyển đổi ngầm, chúng ta có thể sử dụng rõ ràng để ngăn chặn chúng (§16.2.6, §18.4.2).**

**18.3.4 Chữ**

**Chúng tôi có nghĩa đen của các loại được tích hợp sẵn. Ví dụ,1,2 và 12e3 là các loại chữ kép. Vìphức tạp, chúng ta có thể tiến gần đến điều đó bằng cách khai báo các hàm tạo constexpr (§10.4). Ví dụ:**

**lớp phức tạp {**

****

**công cộng:**

**phức hợp constexpr (d**

* ***...***

**}**

**Cho rằng, một phức tạp có thể được xây dựng từ các bộ phận cấu thành của nó tại thời điểm biên dịch giống như một chữ từ một kiểu dựng sẵn. Ví dụ:**

**phức z1 {1,2,12e3};**

**constexpr phức z2 {1,2,12e3}; //*đảm bảo thời gian biên dịch khởi tạo***

**Khi các hàm tạo đơn giản và nội dòng, và đặc biệt là khi chúng constexpr, nó khá hợp lý khi nghĩ về các lời gọi hàm tạo với các đối số theo nghĩa đen là các nghĩa đen.**

**Có thể đi xa hơn và giới thiệu một ký tự do người dùng xác định (§19.2.6) để hỗ trợ Đặc biệt, chúng tôi có thể xác địnhtôi là một hậu tố có nghĩa là '' tưởng tượng. '' Ví dụ:**

**toán tử phức tạp constexpr <double> "" i (d dài kép) {** **// *chữ tưởng tượng***

**trả về {0, d};** **// *phức tạp là một loại chữ***

**}**

****

**Mục 18.3.4** **Chữ viết** **541**

**Điều này sẽ cho phép chúng tôi viết:**

**phức z1 {1,2 + 12e3i};**

**phức f (kép d)**

**{**

**auto x {2.3i};**

**trả về x + sqrt (d + 12e3i) + 12e3i;**

**}**

**Chữ do người dùng xác định này mang lại cho chúng tôi một lợi thế so với những gì chúng tôi nhận được từ constexpr hàm tạo: chúng ta có thể sử dụng các ký tự do người dùng xác định ở giữa các biểu thức trong đó {} ký hiệu chỉ có thể được sử dụng khi đủ điều kiện bởi một loại na**

**phức z1 {1,2,12e3};**

**phức f (kép d)**

**{**

**phức x {0,2.3};**

**return x + sqrt (complex {d, 12e3}) + complex {0,12e3};**

**}**

**Tôi nghi ngờ rằng việc lựa chọn phong cách chữ phụ thuộc vào cảm quan thẩm mỹ của bạn và các quy ước trong lĩnh vực công việc của bạn. Thư viện tiêu chuẩnphức tạp sử dụng constexpr các hàm tạo chứ không phải là một chữ do người dùng xác định.**

**18.3.5 Chức năng của Accessor**

**Cho đến nay, chúng tôi đã cung cấp lớp phức tạp chỉ với các hàm tạo và toán tử số học. Điều đó không hoàn toàn đủ để sử dụng thực tế. Đặc biệt, chúng ta thường cần có khả năng kiểm tra và thay đổi giá trị của phần thực và phần ảo:**



**lớp phức tạp {**

**gấp đôi lại, im;**

**công cộng:**

**constexpr double real () const {return re; }**

**constexpr double image () const {return im; }**

**void real (double r) {re = r; }**

**void hinh (double i) {im =**

**i; } //*...***

**};**

**Tôi không coi việc cung cấp quyền truy cập cá nhân cho tất cả các thành viên của một lớp là một ý tưởng hay; nói chung, nó không phải là. Đối với nhiều loại, quyền truy cập cá nhân (đôi khi được gọi là*hàm get-and-set*) là một lời mời đến thảm họa. Nếu chúng tôi không cẩn thận, quyền truy cập cá nhân có thể làm tổn hại đến một bất biến và nó thường làm phức tạp các thay đổi đối với biểu diễn. Ví dụ: hãy xem xét các cơ hội sử dụng sai mục đích từ việc cung cấp bộ chuyển đổi và bộ định vị cho mọi thành viên củaNgày từ §16.3 hoặc (thậm chí hơn thế) cho Dây từ §19.3. Tuy nhiên, đối vớiphức tạp, thực() và tưởng tượng () có ý nghĩa về mặt ngữ nghĩa: một số thuật toán được viết rõ ràng nhất nếu chúng có thể thiết lập phần thực và phần ảo một cách độc lập.**



**542** **Người vận hành quá tải** **Chương 18**

**Ví dụ, đã cho thực() và tưởng tượng (), chúng tôi có thể đơn giản hóa các hoạt động đơn giản, phổ biến và hữu ích, chẳng hạn như ==, dưới dạng các hàm không phải bộ nhớ (mà không ảnh hưởng đến hiệu suất):**

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

**return a.real () == b.real () && a.imag () == b.imag ();**

**}**

**18.3.6 Chức năng của người trợ giúp**

**Nếu chúng ta đặt tất cả các bit và các mảnh lại với nhau, phức tạp lớp học trở thành:**

**lớp phức tạp {**

**gấp đôi lại, im;**

**công cộng:**

**phức hợp constexpr (double r = 0, double i = 0): re (r), im (i) {}**

**constexpr double real () const {return re; }**

**constexpr double image () const {return im; }**

**void real (double r) {re = r; }**

**void hinh (double i) {im = i; }**

**complex & operator + = (complex);**

**phức & operator + = (double);**

* ***- =, \* =, và / =***

**};**

**Ngoài ra, chúng tôi phải cung cấp một số chức năng trợ giúp:**



**toán tử phức tạp + (phức tạp, phức tạp);**

**toán tử phức tạp + (comp**

**toán tử phức tạp + (awl**

**// *nhị phân -, \* và /***

**toán tử phức hợp− (phức tạp); //*một ngôi trừ***

**toán tử phức tạp + (phức tạp); //*cộng một***

**toán tử bool == (phức tạp, phức tạp);**

**toán tử bool! = (phức tạp, phức tạp);**

**istream & operator >> (istream &, complex &);** **// *đầu vào***

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

**Lưu ý rằng các thành viên thực() và tưởng tượng () rất cần thiết để xác định các so sánh. Các định nghĩa của hầu hết các chức năng trợ giúp sau đây đều dựa vàothực() và tưởng tượng ().**

**Chúng tôi có thể cung cấp các chức năng để cho phép người dùng suy nghĩ về tọa độ cực:**



**Mục 18.3.6** **Chức năng của người trợ giúp** **543**

**cực phức (rho kép, theta kép); liên hợp**

**phức tạp (complex);**

**cơ bụng kép (phức hợp);**

**đôi arg (phức tạp);**

**định mức kép (phức tạp);**

**thực kép (phức);**

**// *để thuận tiện cho việc ghi chú*//**

**hình ảnh kép (phức tạp);**

***để thuận tiện cho việc ghi chú***

**Cuối cùng, chúng ta phải cung cấp một tập hợp các hàm toán học tiêu chuẩn thích hợp:**

**acos phức tạp (phức tạp);**

**asin phức tạp (phức tạp);**

**atan phức tạp (complex);**

* ***...***

**Từ quan điểm của người dùng, phức tạp loại được trình bày ở đây gần giống với phức tạp <double> tìm thấy trong <phức tạp> trong thư viện chuẩn (§5.6.2, §40.4).**

**18.4 Chuyển đổi loại**

**Việc chuyển đổi kiểu có thể được thực hiện bằng**

* **Một hàm tạo nhận một đối số duy nhất (§16.2.5)**
* **Toán tử chuyển đổi (§18.4.1)**

**Trong cả hai trường hợp, chuyển đổi có thể**

* **rõ ràng; nghĩa là, việc chuyển đổi chỉ được thực hiện trong một lần khởi tạo trực tiếp (§16.2.6), tức là, như một bộ khởi tạo không sử dụng=.**
* **Ngụ ý; nghĩa là, nó sẽ được áp dụng ở bất cứ nơi nào nó có thể được sử dụng một cách rõ ràng (§18.4.3), ví dụ, như một đối số của hàm.**



**18.4.1 Các nhà khai thác chuyển đổi**

**Sử dụng một hàm tạo lấy một đối số duy nhất để chỉ định chuyển đổi kiểu là thuận tiện nhưng có những tác động có thể không mong muốn. Ngoài ra, một hàm tạo không thể chỉ định**

1. **chuyển đổi ngầm định từ kiểu do người dùng xác định sang kiểu cài sẵn (vì kiểu cài sẵn không phải là lớp) hoặc**
2. **một chuyển đổi từ một lớp mới sang một lớp đã xác định trước đó (mà không sửa đổi**

**khai báo cho lớp cũ).**

**Những vấn đề này có thể được xử lý bằng cách xác định một *toán tử chuyển đổi* cho loại nguồn. Một chức năng thành viênX :: toán tử T (), ở đâu NS là một tên loại, xác định một chuyển đổi từ NS đến NS. Ví dụ: chúng ta có thể xác định một số nguyên không âm 6 bit,Nhỏ bé, có thể trộn tự do với các số nguyên trong các phép toán số học. Nhỏ bé ném Bad\_range nếu các hoạt động của nó bị tràn hoặc tràn:**



**544** **Người vận hành quá tải** **Chương 18**

**lớp tí hon {**

**char v;**

**void gán (int i) {if (i & ˜077) ném Bad\_range (); v = i; } công**

**cộng:**

**lớp Bad\_range {};**

**Tiny (int i) {giao (i); }**

**Tiny & operator = (int i) {giao (i); trở lại∗cái này; }**

**toán tử int () const {return v; }** **// *chuyển đổi sang hàm int***

**};**

**Phạm vi được kiểm tra khi**

**Không cần kiểm tra phạm vi khi chúng tôi sao chép Nhỏ bé, vì vậy hàm tạo và gán bản sao mặc định là vừa phải.**

**Để bật các phép toán số nguyên thông thường trên Nhỏ bé các biến, chúng tôi xác định chuyển đổi ngầm định từ**

**Nhỏ bé đến NS, Tiny :: toán tử int (). Lưu ý rằng kiểu đang được chuyển đổi thành là một phần của tên toán tử và không thể được lặp lại dưới dạng giá trị trả về của hàm chuyển đổi:**

**Tiny :: operator int () const {return v; } int** **// *đúng***

**Tiny :: operator int () const {return v; }** **// *lỗi***

**Về mặt này, toán tử chuyển đổi cũng giống như một phương thức khởi tạo.**

**Bất cứ khi nào Nhỏ bé xuất hiện nơi một NS là cần thiết, thích hợp NS Được sử dụng. Ví dụ:**

**int main ()**

**{**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tí hon c1** | | **= 2;** |  |
| **Tí hon c2** | | **= 62;** |  |
| **Nhỏ xíu c3 = c2 − c1;** | | | **// *c3 = 60*** |
| **Tí hon c4 = c3;** | | | **// *không kiểm tra phạm vi (không cần thiết)*** |
| **int i = c1 + c2;** | | |  |
| **c1 = c1 + c2;** | | | **// *lỗi phạm vi: c1 không thể là 64*** |
| **i = c3−64;** | | | **// *i = -4*** |
| **c2** | **= c3−64;** | | **// *lỗi phạm vi: c2 không thể là -4*// *không*** |
| **c3** | **= c4;** |  | ***kiểm tra phạm vi (không cần thiết)*** |



**}**

**Các hàm chuyển đổi dường như đặc biệt hữu ích để xử lý cấu trúc dữ liệu khi việc đọc (được thực thi bởi toán tử chuyển đổi) là nhỏ, trong khi việc gán và khởi tạo rõ ràng là ít tầm thường hơn.**

**Các istream và ostream các loại dựa vào một hàm chuyển đổi để kích hoạt các câu lệnh như:**

**while (cin >> x)**

**cout << x;**

**Hoạt động đầu vào cin >> x trả lại một istream &. Giá trị đó được chuyển đổi hoàn toàn thành giá trị biểu thị trạng tháicin. Giá trị này sau đó có thể được kiểm tra bằngtrong khi (xem §38.4.4). Tuy nhiên, nó thường*không phải* một ý tưởng hay để xác định một chuyển đổi ngầm định từ loại này sang loại khác theo cách sao cho**



**Mục 18.4.1** **Nhà điều hành chuyển đổi** **545**

**thông tin bị mất trong quá trình chuyển đổi.**

**Nói chung, điều khôn ngoan là nên tiết kiệm trong việc giới thiệu các nhà khai thác chuyển đổi. Khi được sử dụng quá mức, chúng sẽ dẫn đến sự mơ hồ. Những sự mơ hồ như vậy được trình biên dịch bắt gặp, nhưng chúng có thể gây phiền toái để giải quyết. Có lẽ ý tưởng tốt nhất ban đầu là thực hiện chuyển đổi bằng các hàm được đặt tên, chẳng hạn nhưX :: make\_int (). Nếu một hàm như vậy trở nên đủ phổ biến để làm cho việc sử dụng rõ ràng là không phù hợp, nó có thể được thay thế bằng một toán tử chuyển đổiX :: toán tử int ().**

**Nếu cả chuyển đổi do người dùng xác định và toán tử do người dùng xác định đều được xác định, thì có thể có sự không rõ ràng giữa toán tử do người dùng xác định và toán tử được tích hợp sẵn. Ví dụ:**

**int operator + (Tiny, Tiny);**

**void f (Tiny t, int i) {**

**t + i; //*lỗi, không rõ ràng: '' operator + (t, Tiny (i)) '' hay '' int (t) + i ''?***

**}**

**Do đó, tốt nhất là nên dựa vào chuyển đổi do người dùng xác định hoặc toán tử do người dùng xác định cho một loại nhất định, nhưng không phải cả hai.**

**18.4.2 rõ ràng Nhà điều hành chuyển đổi**

**Các toán tử chuyển đổi có xu hướng được xác định để chúng có thể được sử dụng ở mọi nơi. Tuy nhiên, có thể khai báo toán tử chuyển đổirõ ràng và nó chỉ áp dụng cho việc khởi tạo trực tiếp (§16.2.6), trong đó rõ ràng phương thức khởi tạo sẽ được sử dụng. Ví dụ, thư viện tiêu chuẩn unique\_ptr (§5.2.1, §34.3.1) có chuyển đổi rõ ràng thành bool:**

**mẫu <typename T, typename D = default\_delete <T>>**

**class unique\_ptr {**

**công cộng:**

* ***...***



**toán tử rõ ràng bool () const**

**noexcept; //*...***

**};**

* ***\* điều này có giữ một con trỏ (đó không phải là nullptr) không?***

**Lý do để khai báo toán tử chuyển đổi này rõ ràng là tránh sử dụng nó trong những bối cảnh đáng ngạc nhiên. Xem xét:**

**void use (unique\_ptr <Record> p, unique\_ptr <int> q) {**

**Nếu p)** **// *OK: chúng tôi muốn sử dụng này***

**ném Invalid\_uninque\_ptr {};**

**bool b = p;** **// *lỗi; sử dụng đáng ngờ*// *lỗi; chúng tôi chắc***

**int x = p + q;** ***chắn không muốn điều này***

**}**

**Có unique\_ptrchuyển đổi thành bool không được hợp. Giá trị củaNS sẽ trở thành thật và giá trị của có hợp lệ hay không).**



**rõ ràng, hai định nghĩa cuối cùng sẽ được tổng NS sẽ trở thành 1 hoặc 2 (tuỳ thuộc vào việc NS**

**546** **Người vận hành quá tải** **Chương 18**

**18.4.3 Sự mơ hồ**

**Phép gán giá trị kiểu V đến một đối tượng của lớp NS là hợp pháp nếu có một nhà điều hành chuyển nhượngX ::**

**toán tử = (Z) vậy nên V Là Z hoặc có một chuyển đổi duy nhất về V đến Z. Quá trình khởi tạo được xử lý tương đương.**

**Trong một số trường hợp, giá trị của kiểu mong muốn có thể được xây dựng bằng cách sử dụng lặp lại các hàm tạo hoặc toán tử chuyển đổi. Điều này phải được xử lý bằng các chuyển đổi rõ ràng; chỉ một cấp độ chuyển đổi ngầm định do người dùng xác định là hợp pháp. Trong một số trường hợp, một giá trị của kiểu mong muốn có thể được xây dựng theo nhiều cách; những trường hợp như vậy là bất hợp pháp. Ví dụ:**

**lớp X {/ \* *...* \* / X (int); X (const char∗); };**

**lớp Y {/ \**...* \* / Y (int); }; lớp Z {/ \**...* \* / Z**

**(X); };**

**X f (X);**

**Y f (Y);**

**Z g (Z);**

**void k1 ()**

**{**

**f (1);**

**// *lỗi: không rõ ràng f (X (1)) hoặc f (Y (1))?*//**

**f (X {1});**

***VÂNG***

**f (Y {1});**

**// *VÂNG***

**g ("Mack");**

**// *lỗi: cần hai chuyển đổi do người dùng xác định; g (Z {X {"Mack"}}) không được thử*// *OK:***

**g (X {"Doc"});**

***g (Z {X {"Doc"}})***

**g (Z {"Suzy"});**

**// *Được rồi: g (Z {X {"Suzy"}})***

**}**



**Các chuyển đổi do người dùng xác định chỉ được xem xét nếu không thể giải quyết cuộc gọi mà không có chúng (tức là chỉ sử dụng các chuyển đổi tích hợp sẵn).**

**lớp XX {/ \* *...* \* / XX (int);**

**void h (double);**

**void h (XX);**

**void k2 ()**

**{**

**h (1); //*h (gấp đôi {1}) hay h (XX {1})? h (gấp đôi {1})!***

**}**

**Cuộc gọi h (1) có nghĩa h (gấp đôi (1)) bởi vì giải pháp thay thế đó chỉ sử dụng một chuyển đổi tiêu chuẩn chứ không phải là một chuyển đổi do người dùng xác định (§12.3).**

**Các quy tắc chuyển đổi không phải là quy tắc đơn giản nhất để thực hiện, cũng không phải là tài liệu đơn giản nhất, cũng không phải là quy tắc chung nhất có thể được nghĩ ra. Tuy nhiên, chúng an toàn hơn đáng kể và độ phân giải kết quả thường ít gây ngạc nhiên hơn so với các giải pháp thay thế. Việc giải quyết sự mơ hồ theo cách thủ công sẽ dễ dàng hơn nhiều so với việc tìm ra lỗi do chuyển đổi không đáng ngờ.**



**Mục 18.4.3** **Mơ hồ** **547**

**Sự khăng khăng về phân tích từ dưới lên nghiêm ngặt ngụ ý rằng kiểu trả về không được sử dụng trong giải quyết quá tải. Ví dụ:**

**lớp Bốn {**

**công cộng:**

**Quad (đôi);**

* ***...***

**};**

**Toán tử tứ + (Quad, Quad);**

**void f (double a1, double a {**

**Bộ tứ r1 = a1 + a2; Phần tư r2 = // *thêm dấu chấm động chính xác***

**Phần tư {a1} + a2;** ***kép*// *lực lượng bốn***

**}**

**Lý do cho sự lựa chọn thiết kế này một phần là phân tích từ dưới lên nghiêm ngặt dễ hiểu hơn và một phần là nó không được coi là công việc của trình biên dịch để quyết định độ chính xác mà lập trình viên có thể muốn bổ sung.**

**Khi đã xác định được kiểu của cả hai phía của một lần khởi tạo hoặc phép gán, thì cả hai kiểu đều được sử dụng để giải quyết việc khởi tạo hoặc gán. Ví dụ:**

**lớp Real {**

**công cộng:**

**toán tử double ();**

**toán tử int ();**

* ***...***

**};**

**void g (Real a)**



**{**

**gấp đôi d = a;** **// *d =***

**int i = a;** **// *i = a***

**d = a;** **// *d = a.double ();***

**i = a;** **// *i = a.int ();***

**}**

**Trong những trường hợp này, phân tích kiểu vẫn là từ dưới lên, chỉ với một toán tử duy nhất và các kiểu đối số của nó được xem xét cùng một lúc.**

**18.5 Lời khuyên**

1. **Xác định các toán tử chủ yếu để bắt chước cách sử dụng thông thường; §18.1.**
2. **Xác định lại hoặc cấm sao chép nếu mặc định không phù hợp với một loại; §18.2.2.**
3. **Đối với các toán hạng lớn, sử dụng hăng sô tham chiếu các kiểu đối số; §18.2.4.**

**[4] Để có kết quả lớn, hãy sử dụng hàm tạo di chuyển; §18.2.4.**



**548** **Người vận hành quá tải** **Chương 18**

1. **Ưu tiên các chức năng thành viên hơn các thành viên không phải thành viên đối với các hoạt động cần quyền truy cập vào đại diện; §18.3.1.**
2. **Ưu tiên các hàm không phải thành viên hơn các thành viên cho các hoạt động không cần quyền truy cập vào biểu diễn; §18.3.2.**
3. **Sử dụng không gian tên để liên kết các hàm trợ giúp với lớp '' chúng ''; §18.2.5.**
4. **Sử dụng các hàm không nhớ cho các toán tử đối xứng; §18.3.2.**
5. **Sử dụng các hàm thành viên để thể hiện các toán tử yêu cầu giá trị làm toán hạng bên trái của chúng;**

**§18.3.3.1.**

1. **Sử dụng các ký tự do người dùng xác định để bắt chước ký hiệu thông thường; §18.3.4.**
2. **Cung cấp ''bộ() và hiểu được() các hàm '' chỉ dành cho thành viên dữ liệu nếu ngữ nghĩa cơ bản của một**

**lớp yêu cầu chúng; §1**

1. **Thận trọng với int**
2. **Tránh chuyển đổi phá hủy giá trị ('' thu hẹp ''); §18.4.1.**
3. **Không định nghĩa chuyển đổi giống như cả hàm tạo và toán tử chuyển đổi; §18.4.3.**



**19**



***Tất cả chúng ta đều là những trường hợp đặc biệt.***

***- Albert Camus***

* **Giới thiệu**
* **Nhà điều hành đặc biệt**

**Ghi danh; Gọi chức năng; Hội thảo; Tăng và Giảm; Phân bổ và Giao dịch; Chữ viết do người dùng xác định**

**• Một lớp chuỗi**

**Hoạt động cần thiết; Quyền truy cập vào các ký tự; Đại diện; Chức năng Thành viên; Chức năng Người trợ giúp; Sử dụng chuỗi của chúng tôi**

**• Bạn**

**Tìm bạn bè; Bạn bè và thành viên**



**• Lời khuyên**

**19.1 Giới thiệu**

**Quá tải không chỉ dành cho các phép toán số học và logic. Trên thực tế, các nhà khai thác đóng vai trò quan trọng trong việc thiết kế các thùng chứa (ví dụ:vectơ và bản đồ; §4.4), '' con trỏ thông minh '' (ví dụ:unique\_ptr và shared\_ptr; §5.2.1), trình vòng lặp (§4.5) và các lớp khác liên quan đến quản lý tài nguyên.**

**19.2 Các nhà khai thác đặc biệt**

**Các nhà khai thác**

**[] () -> ++ −− xóa mới**

**chỉ đặc biệt ở chỗ ánh xạ từ việc sử dụng chúng trong mã đến định nghĩa của lập trình viên hơi khác so với ánh xạ được sử dụng cho các toán tử đơn phân và nhị phân thông thường, chẳng hạn như +, <, và ˜ (§18.2.3). Các[] (chỉ số dưới) và () (cuộc gọi) là một trong những toán tử hữu ích nhất do người dùng xác định.**



**550** **Nhà điều hành đặc biệt** **Chương 19**

**19.2.1 Đăng ký**

**Một nhà điều hành[] hàm có thể được sử dụng để cung cấp cho các chỉ số con một ý nghĩa cho các đối tượng lớp. Đối số thứ hai (chỉ số con) của mộtnhà điều hành[] chức năng có thể thuộc bất kỳ loại nào. Điều này làm cho nó có thể xác địnhvectơs, mảng kết hợp, v.v.**

**Ví dụ, chúng ta có thể xác định một kiểu mảng kết hợp đơn giản như sau:**

**struct PGS {**

**vector <cặp <chuỗi, int >> vec; //*vectơ của các cặp {name, value}***

**const int & operator [] (const string &) const; int &**

**operator [] (khuyết điểm**

**};**

**Một PGS giữ một vector của std :: cặpNS. Việc triển khai sử dụng cùng một phương pháp tìm kiếm tầm thường và không hiệu quả như trong §7.7:**

**int & PGS :: operator [] (const string & s)**

* ***tìm kiếm s; trả về một tham chiếu đến giá trị của nó nếu được tìm thấy;*// *nếu không, hãy tạo một cặp {s, 0} mới và trả về một tham chiếu đến giá trị của nó***

**{**

**for (auto x: vec)**

**if (s == x.first) trả về x.second;**

**vec.push\_back ({s, 0});** **// *giá trị ban đầu: 0***

**return vec.back (). thứ hai;** **// *trả về phần tử cuối cùng (§31.2.2)***

**}**

**Chúng ta có thể sử dụng PGS như thế này:**

**int main ()** **// *đếm số lần xuất hiện của mỗi từ trên đầu vào***

**{**



**Các giá trị của PGS;**

**chuỗi buf;**

**while (cin >> buf) ++ giá trị [buf]; cho**

**(tự động x: giá trị.vec)**

**cout << '{' << x.first << ',' << x.second << "} \ n";**

**}**

**Thư viện tiêu chuẩn bản đồ và sơ đồ không có thứ tự là những phát triển tiếp theo của ý tưởng về một mảng kết hợp (§4.4.3, §31.4.3) với các triển khai ít ngây thơ hơn.**

**Một nhà điều hành[]() phải là một người khôngtĩnh chức năng thành viên.**

**19.2.2 Cuộc gọi hàm**

**Lời gọi hàm, nghĩa là, ký hiệu *biểu thức (biểu thức-danh sách),* có thể được hiểu là một phép toán nhị phân với *biểu hiện* như toán hạng bên trái và *danh sách biểu thức* như toán hạng bên phải. Người điều hành cuộc gọi,(), có thể được nạp chồng theo cách giống như các toán tử khác có thể. Ví dụ:**



**Mục 19.2.2** **Gọi hàm** **551**

**struct Hành động {**

**int operator () (int);**

**cặp <int, int> operator () (int,**

**int); toán tử kép () (kép); //*...***

**};**

**void f (Hành động)**

**{**

**int x = act (2);**

**auto y = act (3,4);**

**double z = act (2.3); //**

***...***

**};**

**Danh sách đối số cho một nhà điều hành()() được đánh giá và kiểm tra theo các quy tắc truyền đối số thông thường. Việc ghi đè toán tử cuộc gọi hàm dường như hữu ích chủ yếu để xác định các kiểu chỉ có một thao tác duy nhất và cho các kiểu mà một thao tác chiếm ưu thế. Các*người điều hành cuộc* *gọi* còn được gọi là *nhà điều hành ứng dụng*.**

**Rõ ràng nhất và cũng là quan trọng nhất, việc sử dụng () toán tử là cung cấp cú pháp gọi hàm thông thường cho các đối tượng theo một cách nào đó hoạt động giống như các hàm. Một đối tượng hoạt động giống như một hàm thường được gọi là*đối tượng giống chức năng* hoặc đơn giản là một *đối tượng chức năng* (§3.4.3). Các đối tượng chức năng như**

**vậy cho phép chúng ta viết mã lấy các phép toán tầm thường làm tham số. Trong nhiều trường hợp, điều cần thiết là các đối tượng chức năng có thể chứa dữ liệu cần thiết để thực hiện hoạt động của chúng. có thể xác địnhVímộtdụ,lớpchúngvớimộttôinhà**

**điều hành()() điều đó thêm một giá trị được lưu trữ vào đối số của nó:**

**lớp Thêm {**

**val phức;**

**công cộng:**

**Thêm (phức c): val {c} {}**

**Thêm (gấp đôi r, gấp đôi**

**// *lưu một giá trị***



**void operator () (complex & c) const {c + = val; }**

**// *thêm một giá trị vào đối số***

**};**

**Một đối tượng của lớp Thêm vào được khởi tạo bằng một số phức và khi được gọi bằng cách sử dụng (), nó thêm số đó vào đối số của nó. Ví dụ:**

**void h (vector <complex> & vec, list <complex> & lst, complex z) {**

**for\_each (vec.begin (), vec.end (), Thêm {2,3});**

**for\_each (lst.begin (), lst.end (), Thêm {z});**

**}**

**Điều này sẽ thêm phức tạp {2,3} cho mọi phần tử của vectơ và z cho mọi phần tử của danh sách. Lưu ý rằng Thêm {z} xây dựng một đối tượng được sử dụng lặp đi lặp lại bởi cho mỗi(): Thêm {z}'NS nhà điều hành()() được gọi cho mỗi phần tử của dãy.**

**Tất cả điều này hoạt động bởi vì cho mỗi là một mẫu áp dụng () đến đối số thứ ba của nó mà không quan tâm chính xác đối số thứ ba đó thực sự là gì:**



**552** **Nhà điều hành đặc biệt** **Chương 19**

**mẫu <typename Iter, typename Fct> Fct**

**for\_each (Iter b, Iter e, Fct f)**

**{**

**trong khi (b! = e) f (∗b +**

**+); trả về f;**

**}**

**Thoạt nhìn, kỹ thuật này có vẻ bí truyền, nhưng nó đơn giản, hiệu quả và cực kỳ hữu ích (§3.4.3, §33.4).**

**Lưu ý rằng biểu thức lambda (§3.4.3, §11.4) về cơ bản là một cú pháp để xác định một đối tượng hàm. Ví dụ, chúng tôi đồng**

**void h2 (vectơ <complex> {**

**for\_each (vec.begin (), vec.end (), [] (complex & a) {a + = {2,3};});**

**for\_each (lst.begin (), lst.end (), [] (complex & a) {a + = z;});**

**}**

**Trong trường hợp này, mỗi biểu thức lambda tạo ra tương đương với đối tượng hàm Thêm vào. Các cách sử dụng phổ biến khác củanhà điều hành()() là một toán tử chuỗi con và như một toán tử chỉ số con cho**

**mảng nhiều chiều (§29.2.2, §40.5.2).**

**Một nhà điều hành()() phải là một người khôngtĩnh chức năng thành viên.**

**Các toán tử gọi hàm thường là các mẫu (§29.2.2, §33.5.3).**

**19.2.3 Hội nghị tham khảo**

**Nhà điều hành hội nghị, -> (còn được gọi là *mũi tên* toán tử), có thể được định nghĩa là toán tử hậu tố một ngôi. Ví dụ:**

**lớp Ptr {**

* ***...***



**NS∗ toán tử -> ();**

**};**

**Đối tượng của lớp Ptr có thể được sử dụng để truy cập các thành viên của lớp NS theo cách rất giống với cách con trỏ được sử dụng. Ví dụ:**

**void f (Ptr p)**

**{**

**p−> m = 7;** **// *(p.operator -> ()) -> m = 7***

**}**

**Sự biến đổi của đối tượng P vào con trỏ p.operator -> () không phụ thuộc vào thành viên NSchỉ vào. Đó là ý nghĩa trong đótoán tử -> () là một toán tử hậu tố một ngôi. Tuy nhiên, không có cú pháp mới nào được giới thiệu, vì vậy tên thành viên vẫn được yêu cầu sau->. Ví dụ:**

**void g (Ptr p)**

**{**

**NS∗ q1 = p−>;** **// *lỗi cú pháp***

**NS∗ q2 = p.operator -> ();** **// *VÂNG***

**}**



**Mục 19.2.3** **Tham khảo ý kiến** **553**

**Quá tải -> chủ yếu hữu ích để tạo '' con trỏ thông minh '', nghĩa là, các đối tượng hoạt động giống như con trỏ và ngoài ra thực hiện một số hành động bất cứ khi nào một đối tượng được truy cập thông qua chúng. Thư viện tiêu chuẩn '' con trỏ thông minh ''unique\_ptr và shared\_ptr (§5.2.1) cung cấp nhà điều hành ->. Ví dụ,**

**chúng ta có thể xác định một lớpDisk\_ptr để truy cập các đối tượng được lưu trữ trên đĩa. Disk\_ptrHàm tạo của có một tên có thể được sử dụng để tìm đối tượng trên đĩa, Disk\_ptr :: operator -> () đưa đối tượng vào bộ nhớ chính khi được truy cập thông qua Disk\_ptr, và Disk\_ptrTrình hủy của cuối cùng ghi đối tượng đã cập nhật trở lại đĩa:**

**mẫu <typename T>**

**lớp Disk\_ptr {**

**định danh chuỗi;**

**NS∗ in\_core\_address;**

* ***...***

**công cộng:**

**Disk\_ptr (const string & s): số nhận dạng {s}, in\_core\_address {nullptr} {} ˜Disk\_ptr () {write\_to\_disk (in\_core\_address, Identity); }**

**NS∗ toán tử -> ()**

**{**

**if (in\_core\_address == nullptr)**

**in\_core\_address = read\_from\_disk (mã định danh);**

**trả về địa chỉ in\_core;**

**}**

**};**

**Disk\_ptr có thể được sử dụng như thế này:**

**struct Rec {**

**tên chuỗi;**

* ***...***

**};**



**void update (const string & {**

**Disk\_ptr <Rec> p {s};**

**// *lấy Disk\_ptr cho s***

**p−> name = "Roscoe";**

**// *cập nhật s; nếu cần, trước tiên hãy truy xuất từ đĩa***

**// *...***

**}**

**// *trình hủy của p ghi trở lại đĩa***

**Đương nhiên, một chương trình thực tế sẽ chứa mã xử lý lỗi và sử dụng một cách tương tác ít ngây thơ hơn với đĩa.**

**Đối với con trỏ thông thường, sử dụng -> đồng nghĩa với một số cách sử dụng của một ngôi ∗ và []. Cho một lớp học**

**Y mà ->, ∗, và [] có nghĩa mặc định của chúng và Y∗ gọi là P, sau đó:**

**p−> m == (∗buổi chiều (**

**// *là đúng***

**∗p) .m == p [0] .mp−> m**

**// *là đúng***

**== p [0] .m**

**// *là đúng***

**Như thường lệ, không có sự đảm bảo nào như vậy được cung cấp cho các nhà khai thác do người dùng xác Sự tương đương có thể là định. vided ở nơi mong muốn:**



**554** **Nhà điều hành đặc biệt** **Chương 19**

**mẫu <typename T>**

**lớp Ptr {**

**Y∗ P;**

**công cộng:**

**Y∗ toán tử -> () {return p; } Y &** **// *quyền truy cập thành viên*//**

**nhà điều hành∗() { trở lại ∗P; }** ***dereference để truy cập toàn bộ đối***

**Y & operator [] (int i) {return p** ***tượng*// *quyền truy cập phần tử truy cập***

**[i]; } //*...***

**};**

**Nếu bạn cung cấp nhiều hơn một trong các toán tử này, có thể là khôn ngoan nếu cung cấp các toán tử tương đương, cũng như việc đảm bảo rằng ++NSlớp NS nếu như ++, + =, =, và + là**

**Quá tải của -> là quan trọng đối với một lớp các chương trình thú vị và không chỉ là một sự tò mò nhỏ. Lý do là vậy*sự chuyển hướng* là một khái niệm chính và quá tải -> cung cấp một cách rõ ràng, trực tiếp và hiệu quả để biểu diễn hướng dẫn trong một chương trình. Trình lặp (Chương 33) cung cấp một ví dụ quan trọng về điều này.**

**Nhà điều hành -> phải là một người khôngtĩnh chức năng thành viên. Nếu được sử dụng, kiểu trả về của nó phải là một con trỏ hoặc một đối tượng của một lớp mà bạn có thể áp dụng->. Phần thân của hàm thành viên lớp mẫu chỉ được kiểm tra nếu hàm được sử dụng (§26.2.1), vì vậy chúng ta có thể xác địnhtoán tử -> () mà không cần lo lắng về các loại, chẳng hạn như Ptr <int>, mà -> không có ý nghĩa.**

**Mặc dù sự giống nhau giữa -> và . (dấu chấm), không có cách nào để nạp chồng toán tử . (dấu chấm).**

**19.2.4 Tăng và Giảm**

**Khi mọi người phát minh ra '' con trỏ thông minh '', họ thường quyết định cung cấp toán tử tăng dần ++ và toán tử giảm dần −− để phản ánh việc sử dụng của các toán tử này cho các kiểu cài sẵn. Điều này đặc biệt rõ ràng và cần thiết khi mục đích là thay thế một loại con trỏ thông thường bằng một loại '' con trỏ thông minh '' có cùng ngữ nghĩa, ngoại trừ việc nó thêm một chút kiểm tra lỗi thời gian chạy. Ví dụ, hãy xem xét một giao dịch rắc rối**



**void f1 (X a)**

* ***giao dịch***

**{**

**X v [200];**

**NS∗ p = & v [0];**

**p−−;**

**∗p = a;** **// *oops: p out of range, uncaught***

**+ + p;**

**∗p = a;** **// *VÂNG***

**}**

* **đây, chúng tôi có thể muốn thay thế NS∗ với một đối tượng của một lớp Ptr <X> chỉ có thể được tham chiếu nếu nó thực sự trỏ đến một NS. Chúng tôi cũng muốn đảm bảo rằng P chỉ có thể được tăng và giảm nếu nó trỏ đến một đối tượng trong một mảng và các phép toán tăng và giảm mang lại một đối tượng trong mảng đó. Đó là, chúng tôi muốn một cái gì đó như thế này:**



|  |  |  |
| --- | --- | --- |
| **Mục 19.2.4** | **Tăng và giảm** | **555** |
| **void f2 (Ptr <X> a)** | **// *đã kiểm tra*** |  |

**{**

**X v [200];**

**Ptr <X> p (& v [0], v);**

**p−−;**

**∗p = a;** **// *lỗi thời gian chạy: p ngoài phạm vi***

**+ + p;**

**∗p = a;** **// *VÂNG***

**}**

**Các toán tử tăng và giảm là duy nhất trong số các toán tử C ++ ở chỗ chúng có thể được sử dụng**

**như cả tiền tố và hậu tố**

**và giảm cho Ptr <T>. NS**

**mẫu <typename T>**

**lớp Ptr {**

**NS∗ ptr;**

**NS∗ mảng;**

**int sz;**

**công cộng:**

**mẫu <int N>**

**Ptr (T∗ p, T (& a) [N]);**

**Ptr (T∗ p, T∗ a, int s);**

**Ptr (T∗ P);**

**Ptr & toán tử ++ ();**

**Toán tử ptr ++ (int);**

**Ptr & toán tử −− ();**

**Toán tử ptr −− (int);**

**T & nhà điều hành∗();**

* ***liên kết với mảng a, sz == N, giá trị ban đầu p*// *liên kết với mảng a có kích thước s, giá trị ban đầu p*// *liên kết với một đối tượng, sz == 0, giá trị ban đầu p***
* ***tiếp đầu ngữ***
* ***hậu tố***
* ***tiếp đầu ngữ***
* ***hậu tố***



**};**

**Các NS đối số được sử dụng để chỉ ra rằng hàm sẽ được gọi cho ứng dụng hậu tố của ++. Cái nàyNS không bao giờ được sử dụng; đối số chỉ đơn giản là một giả được sử dụng để phân biệt giữa ứng dụng tiền tố và hậu tố. Cách để nhớ phiên bản nào của mộttoán tử ++ là tiền tố để lưu ý rằng phiên bản không có đối số giả là tiền tố, chính xác như tất cả các toán tử số học và logic một bậc khác. Đối số giả chỉ được sử dụng cho hậu tố '' kỳ lạ ''++ và −−. Cân nhắc bỏ qua hậu tố++ và −− trong một thiết**

**kế. Chúng không chỉ kỳ quặc về mặt cú pháp, chúng có xu hướng khó triển khai hơn một chút so với các phiên bản postfix, kém hiệu quả hơn và ít được sử dụng hơn. Ví dụ:**

|  |  |  |
| --- | --- | --- |
| **mẫu <typename T>** | **// *trả về đối tượng hiện tại sau khi tăng dần*** |  |
| **Ptr & Ptr <T> :: operator ++ () {** |  |

* ***... kiểm tra xem ptr + 1 có thể được trỏ tới ...***

**trở lại ∗++ ptr;**

**}**



|  |  |  |
| --- | --- | --- |
| **556 Nhà điều hành đặc biệt** | **Chương 19** |  |
| **mẫu <typename T>** | **// *tăng và trả về một Ptr với giá trị cũ*** |  |
| **Ptr Ptr <T> :: operator ++ (int) {** |  |

**// *... kiểm tra xem ptr + 1 có thể được trỏ tới ...***

**Ptr <T> cũ {ptr, mảng, sz};**

**+ + ptr;**

**trở lại cũ;**

**}**

**Toán tử tăng trước có thể trả về một tham chiếu đến đối tượng của nó. Toán tử tăng sau phải tạo một đối tượng mới để trả về**

**Sử dụng Ptr, ví dụ tôi**

**void f3 (T a)** **// *đã kiểm tra***

**{**

**T v [200];**

**Ptr <T> p (& v [0], v, 200);**

**p.operator −− (0);** **// *hậu tố: p--***

**p. điều hành viên∗() = a;** **// *lỗi thời gian chạy: p ngoài phạm vi*//**

**p.operator ++ ();** ***tiền tố: ++ p***

**p. điều hành viên∗() = a;** **// *VÂNG***

**}**

**Đang hoàn thành lớp học Ptr được để lại như một bài tập. Một mẫu con trỏ hoạt động chính xác liên quan đến kế thừa được trình bày trong §27.2.2.**

**19.2.5 Phân bổ và Giao dịch**

**Nhà điều hành Mới (§11.2.3) có được bộ nhớ của nó bằng cách gọi một toán tử mới (). Tương tự, toán tửxóa bỏgiải phóng bộ nhớ của nó bằng cách gọi một toán tử xóa (). Người dùng có thể xác định lại toàn cầutoán tử mới () và**



**toán tử xóa () hoặc xác định op**

**Sử dụng các sions chuẩn của thủ thư trông như thế này:**

**vô hiệu∗ toán tử new (size\_t); vô hiệu**

* **toán tử new [] (size\_t); xóa toán tử void (void∗, size\_t); toán tử void xóa [] (void∗, size\_t);**
* ***sử dụng cho từng đối tượng*// *sử dụng cho mảng*// *sử dụng cho từng đối tượng*// *sử dụng cho mảng***
* ***để biết thêm phiên bản, xem §11.2.4***

**Đó là khi Mới cần bộ nhớ trên cửa hàng miễn phí cho một đối tượng thuộc loại NS, Nó gọi toán tử mới (sizeof (X)). Tương tự, khiMới cần bộ nhớ trên cửa hàng miễn phí cho một loạt các n đối tượng của loại NS, Nó gọi toán tử new [] (N∗sizeof (X)). MỘTMới biểu thức có thể yêu cầu nhiều bộ nhớ hơn được chỉ ra bởi n∗sizeof (X), nhưng nó sẽ luôn làm như vậy đối với một số ký tự (tức là một số byte). Thay thế toàn cầu toán tử mới () và toán tử xóa () không dành cho những người yếu đuối và không được khuyến khích. Rốt cuộc, ai đó có thể dựa vào một số khía cạnh của hành vi mặc định hoặc thậm chí có thể đã cung cấp các phiên bản khác của các chức năng này.**



**Mục 19.2.5** **Phân bổ và Giao dịch** **557**

**Một cách tiếp cận chọn lọc hơn và thường tốt hơn là cung cấp các hoạt động này cho một lớp cụ thể. Lớp này có thể là cơ sở cho nhiều lớp dẫn xuất. Ví dụ: chúng tôi có thể muốn có một lớp họcNhân viên cung cấp một trình phân bổ và phân bổ giao dịch chuyên biệt cho chính nó và tất cả các lớp dẫn xuất của nó:**

**nhân viên lớp {**

**công cộng:**

* ***...***

**vô hiệu∗ toán tử new (size\_t); xóa**

**toán tử void (void∗, size\_t);**

**vô hiệu∗ toán tử mới**

**[] xóa toán tử void**

**};**

**Thành viên toán tử mới ()cát toán tử xóa ()s là ngầm tĩnh các thành viên. Do đó, họ không cócái này con trỏ và không sửa đổi một đối tượng. Chúng cung cấp bộ nhớ mà một phương thức khởi tạo có thể khởi tạo và một phương thức hủy có thể dọn dẹp.**

**vô hiệu∗ Nhân viên :: toán tử mới (size\_t s) {**

* ***cấp phát s byte bộ nhớ và trả về một con trỏ tới nó***

**}**

**void Employee :: operator delete (void∗ p, size\_t s) {**

**Nếu p) {** **// *chỉ xóa nếu p! = 0; xem §11.2, §11.2.3***

* ***giả sử p trỏ đến s byte bộ nhớ được cấp phát bởi Employee :: operator new ()*// *và giải phóng bộ nhớ đó để sử dụng lại***

**}**

**}**



**Việc sử dụng đối tượng của tôi cho đến nay xóa bỏNS. Xóa việc xóa mộtNgười quản lý có nguồn gốc từ Nhân viên cái đó không có cái riêng của nó toán tử xóa () cho một giá trị đối số là sizeof (Người quản lý). Điều này cho phép trình phân bổ theo lớp cụ thể tránh lưu trữ thông tin kích thước với mỗi lần phân bổ. Đương nhiên, một trình phân bổ theo lớp cụ thể có thể lưu trữ thông tin như vậy (như một trình phân bổ có mục đích chung phải) và bỏ quasize\_t tranh luận với toán tử xóa (). Tuy nhiên, làm như vậy sẽ khó cải thiện đáng kể tốc độ và mức tiêu thụ bộ nhớ của bộ cấp phát đa năng.**

**Làm thế nào để trình biên dịch biết cách cung cấp kích thước phù hợp cho toán tử xóa ()? Loại được chỉ định trongxóa bỏ hoạt động phù hợp với loại đối tượng xóa bỏNS. Nếu chúng taxóa bỏ một đối tượng thông qua một con trỏ đến một lớp cơ sở, lớp cơ sở đó phải có ảo trình hủy (§17.2.5) cho kích thước chính xác được cung cấp:**

**Nhân viên∗ p = Người quản lý mới; //*rắc rối tiềm ẩn (loại chính xác bị mất)*// *...***

**xóa p;** **// *hy vọng Nhân viên có một trình hủy ảo***

**Về nguyên tắc, việc phân bổ thỏa thuận sau đó được thực hiện bởi hàm hủy (biết kích thước của lớp của nó).**



**558** **Nhà điều hành đặc biệt** **Chương 19**

**19.2.6 Chữ viết do người dùng định nghĩa**

**C ++ cung cấp các ký tự cho nhiều kiểu tích hợp sẵn (§6.2.6):**

1. **// *NS***

**1,2** **// *kép***

**1,2F** **// *flyến mạch***

**'Một'** **// *char***

**1 ĐẦY** **// *dài lâu không dấu*// *hệ thập lục***

**0xD0** ***phân không dấu*// *Chuỗi kiểu C***

**"như"** ***(const char [3])***

**Ngoài ra, chúng ta có thể định**

**nghĩa l Ví dụ:**

|  |  |  |
| --- | --- | --- |
| **"Của anh ấy** | **// *chuỗi, không phải là '' mảng ký tự kết thúc bằng không ''*** | |
| **1,2i** | **// *tưởng tượng*** |  |
| **101010111000101b** | **// *nhị phân*** |  |
| **123 giây** | **// *giây*** |  |
| **123,56km** | **// *không dặm! (các đơn vị)*** |  |
| **1234567890123456789012345678901234567890x** | | **// *độ chính xác mở rộng*** |

**Như là *chữ do người dùng xác định* được hỗ trợ thông qua khái niệm *toán tử chữ* ánh xạ đó các ký tự với một hậu tố nhất định thành một kiểu mong muốn. Tên của toán tử theo nghĩa đen lànhà điều hành"" theo sau là hậu tố. Ví dụ:**

**toán tử phức tạp constexpr <double> "" i (d dài kép) {** **// *chữ tưởng tượng***

**trả về {0, d};** **// *phức tạp là một loại chữ***

**}**

**std :: string operator "" s (const char∗ p, size\_t n) {** **// *std :: chuỗi ký tự***



**trả về chuỗi {p, n};**

**}**

**Hai toán tử này xác định các hậu tố tôi và NS, tương ứng. đánh giá.** **tôi sử dụng constexpr để kích hoạt thời gian biên dịch**

**Với những điều đó, chúng ta có thể viết:**

**mẫu <typename T> void f (const T &);**

**void g ()**

**{**

**f ("Xin chào");** **// *chuyển con trỏ tới ký tự \**// *truyền đối***

**f ("Xin chào" s);** ***tượng chuỗi (năm ký tự)*// *truyền đối***

**f ("Xin chào \ n" s);** ***tượng chuỗi (sáu ký tự)***

**auto z = 2 + 1i;** **// *phức tạp {2,1}***

**}**

* **tưởng (triển khai) cơ bản là sau khi phân tích cú pháp những gì có thể là một chữ, trình biên dịch luôn kiểm tra một hậu tố. Cơ chế chữ do người dùng xác định chỉ đơn giản là cho phép người dùng chỉ định một**



**Mục 19.2.6** **Chữ viết do người dùng xác định** **559**

**hậu tố và xác định những gì sẽ được thực hiện với chữ đứng trước nó. Không thể xác định lại ý nghĩa của một hậu tố theo nghĩa đen có sẵn hoặc để tăng cường cú pháp của các ký tự.**

**Có bốn loại ký tự có thể được thêm hậu tố để tạo thành một ký tự do người dùng xác định (§iso.2.14.8):**

* **Một ký tự số nguyên (§6.2.4.1): được chấp nhận bởi toán tử ký tự lấy một dài lâu không dấu hoặc mộtconst char∗ đối số hoặc bởi một toán tử mẫu, chẳng hạn, 123m hoặc**

**12345678901234567890X**

**• Một ký tự dấu phẩy động (§6.2.5.1): được chấp nhận bởi toán tử ký tự lấy dấu dài đôi hoặc một**

**hăng sô** **char∗** **đối số hoặc bởi một toán tử mẫu, chẳng hạn,**

**12345678901234567890,976543210x hoặc 3,99 giây**

* **Một ký tự chuỗi (§7.3.2): được chấp nhận bởi một toán tử ký tự lấy dấu (const char∗, size\_t) cặp đối số, để kiểm tra**
* **Một ký tự theo nghĩa đen (**

**kiểu char, wchar\_t, char16\_t, hoặc char32\_t, Ví dụ, 'f'\_runic hoặc u'BEEF'\_w.**

**Ví dụ: chúng ta có thể xác định một toán tử chữ để thu thập các chữ số cho các giá trị số nguyên không thể được biểu diễn trong bất kỳ kiểu số nguyên tích hợp nào:**

**Toán tử bignum "" x (const char∗ P) {**

**return Bignum (p);**

**}**

**void f (Bignum);**

**f (123456789012345678901234567890123456789012345x);**

**Đây, chuỗi kiểu C "123456789012345678901234567890123456789012345 " được chuyển cho nhà điều hành""**

**NS(). Lưu ý rằng tôi đã không đặt những chữ số đó trong dấu ngoặc kép. Tôi đã yêu cầu một chuỗi kiểu C cho toán tử của mình và trình biên dịch đã phân phối chuỗi đó từ các chữ số được cung cấp.**

**Để chuyển một chuỗi kiểu C từ văn bản nguồn chương trình thành một toán tử chữ, chúng tôi yêu cầu cả chuỗi và số cha của nó**



**toán tử chuỗi "" s (const**

**string s12 = "một hai" s; string** **// *nhà điều hành cuộc gọi "" ("một hai", 7)*//**

**s22 = "hai \ ntwo" s; string sxx** ***nhà điều hành cuộc gọi "" ("hai \ ntwo", 7)*//**

**= R "(hai \ ntwo)" s;** ***toán tử cuộc gọi "" ("hai \\ ntwo", 8)***

**Trong chuỗi thô (§7.3.2.1), "\n" đại diện cho hai nhân vật '\' và 'n'.**

**Lý do của việc yêu cầu số lượng ký tự là nếu chúng ta muốn có '' một loại chuỗi khác '', chúng ta hầu như luôn muốn biết số lượng ký tự.**

**Một toán tử theo nghĩa đen chỉ cần một const char∗ đối số (và không có kích thước) có thể được áp dụng cho các ký tự số nguyên và dấu phẩy động. Ví dụ:**

**toán tử chuỗi "" SS (const char∗ P);** **// *cảnh báo: điều này sẽ không hoạt động như mong đợi***

**string s12 = "một hai" SS;** **// *lỗi: không có toán tử nghĩa đen áp dụng*//**

**chuỗi s13 = 13SS;** ***OK, nhưng tại sao mọi người lại làm như vậy?***

**Một toán tử chữ chuyển đổi các giá trị số thành chuỗi có thể khá khó hiểu.**



**560** **Nhà điều hành đặc biệt** **Chương 19**

**MỘT *toán tử mẫu* là một toán tử theo nghĩa đen nhận đối số của nó như một gói tham số mẫu, chứ không phải là một đối số hàm. Ví dụ:**

**mẫu <char ...>**

**constexpr int operator "" \_b3 ();** **// *cơ số 3, tức là, bậc ba***

**Do đó, chúng tôi nhận được:**

**201\_b3** **// *nghĩa là toán tử "" b3 <'2', '0', '1'> (); vì vậy 9 \* 2 + 0 \* 3 + 1 == 19*// *nghĩa là toán***

**241\_b3** ***tử "" b3 <'2', '4', '1'> (); vì vậy lỗi: 4 không phải là chữ số bậc ba***

**Các kỹ thuật khuôn mẫu đa dạng (§28.6) có thể gây khó chịu, nhưng đó là cách duy nhất để gán các ý nghĩa không chuẩn cho di**

**Định nghĩa toán tử "" \_b3**

**constexpr int ipow (int x, int n) // *x thành lũy thừa thứ n cho n> = 0*{**

**return (n> 0)? NS∗ipow (n − 1): 1;**

**}**

**mẫu <char c>** **// *xử lý trường hợp một chữ số bậc ba***

**constexpr int b3\_helper () {**

**static\_assert (c <'3', "không phải là chữ số bậc ba");**

**trả lại c;**

**}**

**mẫu <char c, char ... tail> // *bóc một chữ số bậc ba***

**constexpr int b3\_helper () {**

**static\_assert (c <'3', "không phải là chữ số bậc ba"); return ipow (3,**

**sizeof ... (tail))∗(c − '0 ') + b3\_helper (tail ...);**



**}**

**Do đó, chúng ta có thể xác định ou**

**template <char ... chars>**

**constexpr int operator "" \_b3 () {** **// *cơ số 3, tức là, bậc ba***

**return b3\_helper (ký tự ...);**

**}**

**Nhiều hậu tố sẽ ngắn (ví dụ: NS vì std :: string, tôi cho tưởng tượng, NS cho mét (§28.7.3), và NS để mở rộng), vì vậy các mục đích sử dụng khác nhau có thể dễ dàng xung đột. Sử dụng không gian tên để ngăn đụng độ:**

**không gian tên Numerics {**

* ***...***

**lớp Bignum {/ \* *...* \* /};**

**ký tự không gian tên {**

**Toán tử bignum "" x (char const∗);**

**}**



**Mục 19.2.6** **Chữ viết do người dùng xác định** **561**

* ***...***

**}**

**sử dụng không gian tên Numerics :: chữ;**

**Thư viện tiêu chuẩn dự trữ tất cả các hậu tố không bắt đầu bằng dấu gạch dưới ban đầu, vì vậy hãy xác định các hậu tố của bạn bắt đầu bằng dấu gạch dưới hoặc có nguy cơ bị hỏng mã trong tương lai:**

|  |  |
| --- | --- |
| **123km** | **// *được đặt trước bởi thư viện tiêu chuẩn*//** |
| **123\_km** | ***có sẵn để bạn sử dụng*** |

**19.3 Một lớp chuỗi**

**Lớp chuỗi tương đối đơn giản được trình bày trong phần này minh họa một số kỹ thuật hữu ích cho việc thiết kế và triển khai các lớp bằng cách sử dụng các toán tử được định nghĩa theo quy ước. Cái nàyDây là một phiên bản đơn giản của thư viện tiêu chuẩn dây (§4.2, Chương 36). Dây cung cấp ngữ nghĩa giá trị, quyền truy cập đã kiểm tra và không kiểm tra vào các ký tự, luồng I / O, hỗ trợ cho phạm vi-vì vòng lặp, phép toán đẳng thức và toán tử nối. Tôi cũng đã thêm mộtDây nghĩa đen, mà std :: stringkhông (chưa) có.**

**Để cho phép khả năng tương tác đơn giản với các chuỗi kiểu C (bao gồm các ký tự chuỗi (§7.3.2)), tôi biểu diễn các chuỗi dưới dạng các mảng ký tự có kết thúc bằng không. Đối với chủ nghĩa hiện thực, tôi thực hiện*tối ưu hóa* *chuỗi ngắn*. Đó là mộtDâychỉ với một vài ký tự lưu trữ các ký tự đó trong chính đối tượng lớp, thay vì trên cửahàng miễn phí. Điều này tối ưu hóa việc sử dụng chuỗi cho các chuỗi nhỏ. Kinh nghiệm cho thấy rằng đối với một số lượng lớn các ứng dụng, hầu hết các chuỗi đều ngắn. Việc tối ưu hóa này đặc biệt quan trọng trong các hệ thống đa luồng nơi mà việc chia sẻ thông qua con trỏ (hoặc tham chiếu) là không khả thi và việc phân bổ và phân bổ kho lưu trữ miễn phí tương đối tốn kém.**



**Cho phép Dâyđể '' phát triển '' một cách hiệu quả bằng cách thêm các ký tự vào cuối, tôi triển khai một kế hoạch để giữ thêm không gian cho**

**Dây một mục tiêu thích hợp cho v**

**Viết một lớp chuỗi tốt hơn và / hoặc một lớp cung cấp nhiều tiện ích hơn là một bài tập tốt. Làm xong, chúng ta có thể vứt bỏ các bài tập của mình và sử dụngstd :: string (Chương 36).**

**19.3.1 Các hoạt động cần thiết**

**Lớp Dây cung cấp tập hợp thông thường của các hàm tạo, một hàm hủy và các hoạt động gán (§17.1):**

**chuỗi lớp {**

**công cộng:**

**Dây();**

**// *hàm tạo mặc định: x {""}***

**Chuỗi rõ ràng (const char∗ P);**

**// *phương thức khởi tạo từ chuỗi kiểu C: x {"Euler"}***

**Chuỗi (const String &);**

**// *sao chép hàm tạo***

**Chuỗi & toán tử = (const Chuỗi &);**

**// *sao chép bài tập***



**562** **Nhà điều hành đặc biệt** **Chương 19**

**Chuỗi (Chuỗi && x);** **// *di chuyển hàm tạo***

**Chuỗi & toán tử = (Chuỗi && x);** **// *chuyển nhiệm vụ***

**˜String () {if (short\_max <sz) delete [] ptr; }** **// *người phá hủy***

**// *...***

**};**

**Cái này Dây có giá trị ngữ nghĩa. Đó là, sau một nhiệm vụs1 = s2, hai chuỗi s1 và s2 hoàn toàn khác biệt và những thay đổi tiếp theo đối với cái này không ảnh hưởng đến cái khác. Sự thay thế sẽ là cung cấp choDây ngữ nghĩa con trỏ. Điều đó sẽ là để cho các thay đổi đối vớis2 sau s1 = s2 cũng ảnh hưởng đến giá trị của s1. Nơi nó tạo ra sendây. Tuy nhiên, đối với giá trị**

**chúng tôi không cần bản sao và triển khai ngữ nghĩa di chuyển (§3.3.2, §17.5.2) để tối ưu hóa trở lạiNS. Đại diện hơi tầm thường củaDây được trình bày trong §19.3.3. Lưu ý rằng nó yêu cầu các phiên bản do người dùng xác**

**định của các hoạt động sao chép và di chuyển.**

**19.3.2 Quyền truy cập vào các ký tự**

**Việc thiết kế các toán tử truy cập cho một chuỗi là một chủ đề khó vì truy cập lý tưởng là bằng ký hiệu thông thường (nghĩa là sử dụng []), hiệu quả tối đa và đã kiểm tra phạm vi. Thật không may, bạn không thể có tất cả các thuộc tính này cùng một lúc. Ở đây, tôi tuân theo thư viện tiêu chuẩn bằng cách cung cấp các hoạt động không được kiểm soát hiệu quả với[] ký hiệu chỉ số dưới cộng với dải ô đã kiểm tra tại()hoạt động:**

**chuỗi lớp {**

**công cộng:**

**// *...***

**char & operator [] (int n) {return ptr [n]; }** **// *quyền truy cập phần tử không được kiểm tra***



**toán tử char [] (int n**

**char & at (int n) {chec**

**char at (int n) const {check (n); return ptr [n]; }**

**Chuỗi & toán tử + = (char c);** **// *thêm c vào cuối***

**const char∗ c\_str () {return ptr; } const** **// *Truy cập chuỗi kiểu C***

**char∗ c\_str () const {return ptr; }**

**int size () const {return sz; } int** **// *số phần tử*// *các yếu tố cộng với***

**dung lượng () const** ***không gian có sẵn***

**{return (sz <= short\_max)? short\_max: sz + dấu cách; }**

* ***...***

**};**

**Ý tưởng là sử dụng [] để sử dụng thông thường. Ví dụ:**



**Mục 19.3.2** **Quyền truy cập vào các ký tự** **563**

**int hash (const String & s) {**

**int h {s [0]};**

**for (int i {1}; i! = s.size (); i ++) h ˆ = s [i] >> 1; trả lại h;**

* ***quyền truy cập không được chọn vào s***

**}**

**Ở đây, bằng cách sử dụng tại() sẽ là dư thừa vì chúng tôi truy cập chính xác NS chỉ từ 0 đến s.size () - 1.**

**Chúng ta có thể sử dụng tại() nơi chúng tôi thấy có thể xảy ra sai lầm. Ví dụ:**

**void print\_in\_order (const {**

**for (x: index) cout << s.at (x) << '\ n';**

**}**

**Thật không may, giả sử rằng mọi người sẽ sử dụng tại() thường xuyên ở những nơi có thể mắc sai lầm là lạc quan quá mức, vì vậy một số cách triển khai std :: string (từ đó []/tại() quy ước là mượn) cũng kiểm tra []. Cá nhân tôi thích kiểm tra hơn[] ít nhất là trong quá trình phát triển. Tuy nhiên, đối với các tác vụ thao tác chuỗi nghiêm trọng, việc kiểm tra phạm vi đối với mỗi quyền truy cập ký tự có thể áp đặt chi phí khá đáng chú ý.**

**tôi cung cấp hăng sô và khônghăng sô phiên bản của các chức năng truy cập để cho phép chúng được sử dụng cho**

**hăng sô cũng như các đối tượng khác.**

**19.3.3 Trình bày**

**Đại diện cho Dây đã được chọn để đáp ứng ba mục tiêu:**

* **Để dễ dàng chuyển đổi một chuỗi kiểu C (ví dụ: một chuỗi ký tự) thành Dây và cho phép dễ dàng truy cập vào các ký tự của một Dây như một chuỗi kiểu C**
* **Để giảm thiểu việc sử dụng cửa hàng miễn phí**
* **Để thêm ký tự**



**Kết quả rõ ràng là lộn xộn hơn**

**chuỗi lớp {**

**/ \***

***Một chuỗi đơn giản thực hiện tối ưu hóa chuỗi ngắn***

***size () == sz là số phần tử***

***if size () <= short\_max, các ký tự được giữ trong chính đối tượng String; nếu không thì cửa hàng miễn phí được sử dụng.***

***ptr trỏ đến đầu chuỗi ký tự***

***dãy ký tự được giữ nguyên bằng không: ptr [size ()] == 0;***

***điều này cho phép chúng tôi sử dụng các hàm chuỗi thư viện C và dễ dàng trả về một chuỗi kiểu C: c\_str ()***

***Để cho phép bổ sung hiệu quả các ký tự vào cuối, Chuỗi phát triển bằng cách tăng gấp đôi phân bổ của nó; dung lượng () là lượng không gian có sẵn cho các ký tự (không bao gồm số 0 kết thúc): sz + dấu cách***

**\* /**



**564** **Nhà điều hành đặc biệt** **Chương 19**

**công cộng:**

* ***...***

**riêng:**

**static const int short\_max = 15;**

**int sz;** **// *số ký tự***

**char∗ ptr;**

**liên hiệp {**

**không gian int;**

**// *không gian được phân bổ không sử dụng*//**

**char ch [short\_max + 1];**

***để lại khoảng trống để kết thúc 0***

**};**

**void check (int n) con {**

**nếu (n <0 || sz <= n)**

**ném std :: out\_of\_range ("String :: at ()");**

**}**

* ***chức năng thành viên phụ trợ:* void copy\_from (const String & x); void move\_from (Chuỗi & x);**

**};**

**Điều này hỗ trợ những gì được gọi là *tối ưu hóa chuỗi ngắn* bằng cách sử dụng hai biểu diễn chuỗi:**

* **Nếu như sz <= short\_max, các ký tự được lưu trữ trong Dây chính đối tượng, trong mảng có tên ch.**
* **Nếu như ! (sz <= short\_max), các ký tự được lưu trữ trên cửa hàng miễn phí và chúng tôi có thể cấp thêm không gian để mở rộng. Thành viên có tênkhông gian là số ký tự như vậy.**

**Trong cả hai trường hợp, số phần tử được giữ trong sz và chúng tôi nhìn vào sz, để xác định lược đồ triển khai nào được sử dụng cho một chuỗi nhất định.**



**Trong cả hai trường hợp, ptr trỏ đến các phần tử. Điều này là cần thiết cho hiệu suất: các chức năng truy cập không cần phải kiểm tra**

**ments, di chuyển và phá hủy**

**Chúng tôi sử dụng mảng ch chỉ nếu sz <= short\_max và số nguyên không gian chỉ nếu ! (sz <= short\_max). Con-**

**do đó, sẽ rất lãng phí nếu phân bổ không gian cho cả hai ch và không gian trong một Dây sự vật. Để tránh lãng phí như vậy, tôi sử dụngliên hiệp (§8.3). Đặc biệt, tôi đã sử dụng một hình thứcliên hiệp được gọi là *công đoàn ẩn danh*(§8.3.2), được thiết kế đặc biệt để cho phép một lớp quản lý các biểu diễn thay thế của các đối tượng. Tất cả các thành viên của một liên minh ẩn danh được cấp phát trong cùng một bộ nhớ, bắt đầu tại cùng một địa chỉ. Chỉ một thành viên có thể được sử dụng tại bất kỳ thời điểm nào, nhưng nếu không chúng được truy cập và sử dụng chính xác như thể chúng là các thành viên riêng biệt của phạm vi xung quanh liên minh ẩn danh. Nhiệm vụ của lập trình viên là đảm bảo rằng chúng không bao giờ bị lạm dụng. Ví dụ, tất cả các chức năng thành viên củaDây sử dụng không gian phải chắc chắn rằng nó thực sự là không gian điều đó đã được thiết lập và không ch. Điều đó được thực hiện bằng cách nhìn vàosz <= short\_max. Nói cách khác,Hình dạng là (trong số những thứ khác) một liên minh bị phân biệt đối xử với sz <= short\_max với tư cách là người phân biệt.**



**Mục 19.3.3.1** **Các chức năng phụ trợ** **565**

**19.3.3.1 Các chức năng phụ trợ**

**Ngoài các chức năng dành cho mục đích sử dụng chung, tôi nhận thấy rằng mã của mình trở nên sạch hơn khi tôi cung cấp ba hàm phụ trợ là '' khối xây dựng '' để giúp tôi biểu diễn hơi phức tạp và giảm thiểu việc sao chép mã. Hai trong số những người cần truy cập vào đại diện củaDây, vì vậy tôi đã biến họ thành thành viên. Tuy nhiên, tôi đã làm cho chúngriêng thành viên bởi vì họ không đại diện cho các hoạt động thường hữu ích và an toàn để sử dụng. Đối với nhiều lớp thú vị, việc triển khai không chỉ là biểu diễn cộng vớicông cộng chức năng. Các chức năng phụ trợ có thể dẫn đến ít trùng lặp mã hơn, thiết kế tốt hơn và cải thiện khả năng bảo trì.**

**Hàm đầu tiên như vậy di chuyển các ký tự vào bộ nhớ mới được cấp phát:**

**char∗ mở rộng (const char∗{**

|  |  |
| --- | --- |
| **char∗ p = new char [n];** |  |
| **strcpy (p, ptr);** | **// *§43.4*** |

**trả lại p;**

**}**

**Chức năng này không truy cập vào Dây đại diện, vì vậy tôi đã không làm cho nó trở thành một thành viên.**

**Chức năng triển khai thứ hai được sử dụng bởi các hoạt động sao chép để cung cấp Dây một bản sao của các thành viên khác:**

**void String :: copy\_from (const String & x)**

**// *làm cho \* đây là một bản sao của x***

**{**

**if (x.sz <= short\_max) {** **// *sao chép cái này***

**memcpy (this, & x, sizeof (x));** **// *§43.5***

**ptr = ch;**

**}**

**khác {** **// *sao chép các yếu tố***

**ptr = expand (x.ptr, x.sz + 1);**



**sz = x.sz;**

**dấu cách = 0;**

**}**

**}**

**Mọi hoạt động dọn dẹp mục tiêu cần thiết Dây là nhiệm vụ của những người gọi copy\_from (); copy\_from () ghi đè mục tiêu của nó một cách vô điều kiện. Tôi sử dụng thư viện tiêu chuẩnmemcpy () (§43.5) để sao chép các byte của nguồn vào đích. Đó là một chức năng cấp thấp và đôi khi khá khó chịu. Nó chỉ nên được sử dụng khi không có đối tượng nào có hàm tạo hoặc hàm hủy trong bộ nhớ đã sao chép vìmemcpy () không biết gì về các loại. Cả haiDây sao chép hoạt động sử dụng copy\_from (). Chức năng tương ứng cho các hoạt động di**

**chuyển là:**

**void String :: move\_from (Chuỗi & x) {**

**if (x.sz <= short\_max) {** **// *sao chép cái này***

**memcpy (this, & x, sizeof (x));** **// *§43.5***

**ptr = ch;**

**}**



**566** **Nhà điều hành đặc biệt** **Chương 19**

**khác {** **// *lấy các yếu tố***

**ptr = x.ptr;**

**sz = x.sz;**

**dấu cách = x.space;**

**x.ptr = x.ch;** **// *x = ""***

**x.sz = 0;**

**x.ch [0] = 0;**

**}**

**}**

**Nó quá vô điều kiện làm cho mục tiêu của nó trở thành một bản sao của lập luận của nó. Tuy nhiên, nó không để lại lập luận của mình sở hữu bất kỳ cửa hàng miễn phí nào**

**biểu diễn chuỗi dài u**

**bers riêng lẻ.**

**19.3.4 Chức năng thành viên**

**Hàm tạo mặc định xác định một Dây để trống:**

**Chuỗi :: Chuỗi ()** **// *hàm tạo mặc định: x {""}*// *ptr trỏ đến các phần tử, ch***

**: sz {0}, ptr {ch}** ***là vị trí ban đầu (§19.3.3)***

**{**

**ch [0] = 0;** **// *chấm dứt 0***

**}**

**Được cho copy\_from () và move\_from (), hàm tạo, di chuyển và phép gán khá đơn giản để thực hiện. Hàm tạo nhận đối số chuỗi kiểu C phải xác định số ký tự và lưu trữ chúng một cách thích hợp:**

**Chuỗi :: Chuỗi (const char∗ p):**

**sz {strlen (p)},**

**ptr {(sz <= short\_max)? ch: new char [sz + 1]}, dấu**



**cách {0}**

**{**

**strcpy (ptr, p); //*sao chép các ký tự vào ptr từ p***

**}**

**Nếu đối số là một chuỗi ngắn, ptr được thiết lập để trỏ đến ch; nếu không, không gian được phân bổ trên cửa hàng miễn phí. Trong cả hai trường hợp, các ký tự được sao chép từ chuỗi đối số vào bộ nhớ được quản lý**

**qua Dây.**

**Hàm tạo sao chép chỉ cần sao chép biểu diễn của các đối số của nó:**

**Chuỗi :: Chuỗi (const String & x) {** **// *sao chép hàm tạo***

**copy\_from (x); //*sao chép đại diện từ x***

**}**

**Tôi không bận tâm đến việc cố gắng tối ưu hóa trường hợp kích thước của nguồn bằng với kích thước của mục tiêu (như đã được thực hiện cho vectơ; §13.6.3). Tôi không biết liệu điều đó có đáng giá hay không.**

**Tương tự, hàm tạo di chuyển di chuyển biểu diễn từ nguồn của nó (và có thể đặt đối số của nó là chuỗi trống):**



|  |  |  |
| --- | --- | --- |
| **Mục 19.3.4** | **Chức năng thành viên** | **567** |
| **Chuỗi :: Chuỗi (Chuỗi && x) {** | **// *di chuyển hàm tạo*** |  |
| **move\_from (x);** |  |  |
| **}** |  |  |

**Giống như hàm tạo bản sao, phép gán bản sao sử dụng copy\_from () để sao chép biểu diễn của đối số của nó. Ngoài ra, nó phảixóa bỏ bất kỳ cửa hàng miễn phí nào thuộc sở hữu của mục tiêu và đảm bảo rằng nó không gặp rắc rối với việc tự chuyển nhượng (ví dụ: s = s):**

**Chuỗi & Chuỗi :: operator = (const String & x) {**

**if (this == & x) return ∗**

**char∗ p = (short\_ma**

**copy\_from (x);**

**xóa [] p;**

**trở lại ∗cái này;**

**}**

**Các Dây chuyển nhiệm vụ sẽ xóa cửa hàng miễn phí của mục tiêu (nếu có) và sau đó di chuyển:**

**Chuỗi & Chuỗi :: operator = (Chuỗi && x) {**

**if (this == & x) return ∗cái này;** **// *đối phó với việc tự giao (x = move (x) là điên rồ)*//**

**if (short\_max <sz) xóa [] ptr;** ***xóa mục tiêu***

**move\_from (x);** **// *không ném***

**trở lại ∗cái này;**

**}**

**Về mặt logic, có thể di chuyển một nguồn vào chính nó (ví dụ: s = std :: move (s)), vì vậy một lần nữa chúng ta phải bảo vệ chống lại sự tự chỉ định (tuy nhiên khó xảy ra).**

**Phức tạp nhất về mặt logic Dây hoạt động là + =, thêm một ký tự vào cuối**

**chuỗi, tăng kích thước của nó lên một:**



**Chuỗi & Chuỗi :: toán tử + {**

**if (sz == short\_max) {** **// *mở rộng thành chuỗi dài*// *tăng gấp đôi phân bổ (+2 vì***

**int n = sz + sz + 2; ptr = *số 0 kết thúc)***

**expand (ptr, n);**

**không gian = n − sz − 2;**

**}**

**khác nếu (short\_max <sz) {**

**if (dấu cách == 0) {** **// *mở rộng trong cửa hàng miễn phí*// *tăng gấp đôi phân bổ***

**int n = sz + sz + 2; char∗*(+2*p =*vì số 0 kết thúc)***

**expand (ptr, n); xóa [] ptr;**

**ptr = p;**

**không gian = n − sz − 2;**

**}**

**khác**

**- - khoảng trống;**

**}**



**568** **Nhà điều hành đặc biệt** **Chương 19**

**ptr [sz] = c;** **// *thêm c vào cuối***

**ptr [++ sz] = 0;** **// *tăng kích thước và đặt dấu chấm hết***

**trở lại ∗cái này;**

**}**

**Có rất nhiều điều đang diễn ra ở đây: toán tử + = () phải theo dõi xem biểu diễn nào (ngắn hay dài) được sử dụng và liệu có thêm không gian để mở rộng hay không. Nếu cần thêm dung lượng,mở rộng() được gọi để phân bổ không gian đó và di chuyển các ký tự cũ vào không gian mới. Nếu có một phân bổ cũ cần xóa**

**có thể, nó là tầm thường để đặt n**

**Lưu ý tính toán bộ nhớ khả dụng cho không gian. Trong tất cả cácDây quá trình triển khai mất nhiều thời gian nhất để làm đúng: một phép tính nhỏ lộn xộn dễ xảy ra lỗi từng lỗi một. Hằng số lặp đi lặp lại đó2 cảm thấy khủng khiếp giống như một '' hằng số ma thuật. ''**

**Tất cả các Dây các thành viên cẩn thận không sửa đổi một đại diện mới trước khi họ chắc chắn rằng một đại diện mới có thể được đưa vào. Đặc biệt, họ khôngxóa bỏ cho đến sau khi có thể Mới các hoạt động đã được thực hiện. Trên thực tếDây các thành viên cung cấp bảo đảm ngoại lệ mạnh mẽ (§13.2).**

**Nếu bạn không thích loại mã khó hiểu được trình bày như một phần của việc triển khai Dây, sim-**

**sử dụng ply std :: string. Ở một mức độ lớn, các cơ sở thư viện tiêu chuẩn tồn tại để giúp chúng ta không phải lập trình ở mức thấp này hầu hết thời gian. Mạnh hơn: viết một lớp chuỗi, một lớp vectơ hoặc một bản đồ là một bài tập tuyệt vời. Tuy nhiên, một khi bài tập được thực hiện, một kết quả sẽ là sự đánh giá cao những gì tiêu chuẩn mang lại và không mong muốn duy trì phiên bản của riêng bạn.**

**19.3.5 Chức năng của người trợ giúp**

**Để hoàn thành lớp học Dây, Tôi cung cấp một tập hợp các chức năng hữu ích, luồng I / O, hỗ trợ cho phạm vi-vì vòng lặp, so sánh và nối. Tất cả những điều này phản ánh các lựa chọn thiết kế được sử dụng chostd :: string. Đặc biệt,<< chỉ in cách đọc chara cho đến khi nó tìm thấy điểm cuối**



**ostream & operator << (ostream & os, const String & s) {**

|  |  |
| --- | --- |
| **trả về os << s.c\_str ();** | **// *§36.3.3*** |

**}**

**istream & operator >> (istream & is, String & s) {**

**s = "";** **// *xóa chuỗi mục tiêu*// *bỏ qua***

**là >> ws;** ***khoảng trắng (§38.4.5.1)***

**char ch = '';**

**while (is.get (ch) &&! isspace (ch))**

**s + = ch;**

**trở lại là;**

**}**

**tôi cung cấp == và ! = để so sánh:**



**Mục 19.3.5** **Chức năng của người trợ giúp** **569**

**toán tử bool == (const String & a, const String & b) {**

**if (a.size ()! = b.size ())**

**trả về sai;**

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

**if (a [i]! = b [i])**

**trả về sai;**

**trả về true;**

**}**

**toán tử bool! = (const Str {**

**return! (a == b);**

**}**

**Thêm <, v.v., sẽ là tầm thường. Để hỗ trợ phạm vi-vì vòng lặp, chúng tôi cần bắt đầu() và kết thúc() (§9.5.1). Một lần nữa, chúng tôi có thể cung cấp các chức năng đó dưới dạng các chức năng tự do (không phải phần tử) mà không cần truy cập trực tiếp vàoDây**

**thực hiện:**

**char∗ bắt đầu (Chuỗi & x) {** **// *Truy cập kiểu C-string***

**trả về x.c\_str ();**

**}**

**char∗ end (Chuỗi & x) {**

**return x.c\_str () + x.size ();**

**}**

**const char∗ begin (const String & x) {**



**trả về x.c\_str ();**

**}**

**const char∗ end (const String & x) {**

**return x.c\_str () + x.size ();**

**}**

**Cho chức năng thành viên + = thêm một ký tự vào cuối, các toán tử nối dễ dàng được cung cấp dưới dạng các hàm không phải là ký tự:**

**Chuỗi & toán tử + = (Chuỗi & a, const String & b) {** **// *nối***

**for (auto x: b)**

**a + = x;**

**trả lại a;**

**}**



**570** **Nhà điều hành đặc biệt** **Chương 19**

**Toán tử chuỗi + (const String & a, const String & b) // *nối*{**

**Chuỗi res {a};**

**res + = b;**

**trả lại res;**

**}**

**Tôi cảm thấy rằng tôi có thể đã hơi '' lừa dối '' ở đây. Tôi có nên cung cấp một thành viên không+ = đã thêm một chuỗi Cstyle vào cuối? Thư viện tiêu chuẩndây nhưng không có nó, nối với chuỗi kiểu C vẫn hoạt động. Ví dụ:**

**Chuỗi s = "Njal"; s + = "Gunnar";**

**// *lừa đảo***

**Việc sử dụng này của + = được hiểu là toán tử + = (s, String ("Gunnar")). Tôi đoán là tôi có thể cung cấp một Chuỗi :: operator + = (const char∗), nhưng tôi không biết liệu hiệu suất được bổ sung có đáng giá trong mã thế giới thực hay không. Trong những trường hợp như vậy, tôi cố gắng tiết chế và đưa ra thiết kế tối giản. Có thể làm một điều gì đó không phải là một lý do chính đáng để làm điều đó.**

**Tương tự, tôi không cố gắng tối ưu hóa + = bằng cách tính đến kích thước của một chuỗi nguồn. Thêm\_NS như một chuỗi nghĩa là hậu tố theo nghĩa đen Dây là tầm thường:**

**Toán tử chuỗi "" \_s (const char∗ p, size\_t) {**

**return Chuỗi {p};**

**}**

**Bây giờ chúng ta có thể viết:**

**void f (const char∗);**

**// *Chuỗi kiểu C***

**void f (const String &);**

**// *chuỗi của chúng tôi***

**void g ()**



**{**

**f ("Của Madden");**

**f (\_s "của Christopher)**

**}**

**19.3.6 Sử dụng chuỗi của chúng tôi**

**Chương trình chính chỉ đơn giản thực hiện Dây toán tử một chút:**

**int main ()**

**{**

**Chuỗi s ("abcdefghij");**

**cout << s << '\ n';**

**s + = 'k';**

**s + = 'l';**

**s + = 'm';**

**s + = 'n';**

**cout << s << '\ n';**



**Mục 19.3.6** **Sử dụng chuỗi của chúng tôi** **571**

**String s2 = "Địa ngục";**

**s2 + = "và nước cao";**

**cout << s2 << '\ n';**

**Chuỗi s3 = "qwerty";**

**s3 = s3;**

**String s4 = "con cáo nâu nhanh nhẹn nhảy qua con chó lười biếng"; s4 = s4;**

**cout << s3 << "" << s4 << "\ n";**

**cout << s + "." + s3 + String (".") + "Horsefeathers \ n";**

**Chuỗi buf;**

**while (cin >> buf && b**

**cout << buf << "" << buf.size () << "" << buf.capacity () << '\ n';**

**}**

**Cái này Dây thiếu nhiều tính năng mà bạn có thể coi là quan trọng hoặc thậm chí cần thiết. Tuy nhiên, đối với những gì nó làm thì nó gần giống vớistd :: string (Chương 36) và minh họa các kỹ thuật được sử dụng để triển khai thư viện tiêu chuẩn dây.**

**19.4 Bạn bè**

**Một khai báo hàm thành viên bình thường chỉ định ba điều khác biệt về mặt logic:**

1. **Hàm có thể truy cập phần riêng của khai báo lớp.**
2. **Hàm thuộc phạm vi của lớp.**
3. **Hàm phải được gọi trên một đối tượng (có cái này con trỏ).**

**Bằng cách khai báo một chức năng thành viên tĩnh (§16.2.12), chúng ta chỉ có thể cung cấp cho nó hai thuộc tính đầu tiên. Bằng cách khai báo một hàm nonmember abạn bè, chúng tôi chỉ có thể cung cấp cho nó tài sản đầu tiên. Đó là, một hàm được khai báobạn bè được cấp quyền truy cập vào việc triển khai một lớp giống như một hàm thành viên nhưng độc lập với th**



**Ví dụ, chúng tôi có thể và Ma trận ẩn các biểu diễn tương ứng của chúng và cung cấp một tập hợp các thao tác hoàn chỉnh để thao tác các đối tượng cùng loại. Tuy nhiên, thói quen nhân của chúng ta không thể là thành viên của cả hai. Ngoài ra, chúng tôi không thực sự muốn cung cấp các chức năng truy cập cấp thấp để cho phép mọi người dùng vừa đọc và viết bản trình bày hoàn chỉnh của cả haiMa trận và Véc tơ. Để tránh điều này, chúng tôi tuyên bốoper-**

**ator∗ Một bạn bè của cả hai:**

**constexpr rc\_max {4};** **// *kích thước hàng và cột***

**Ma trận lớp học;**

**Vectơ lớp {**

**float v [rc\_max];**

* ***...***

**bạn bè toán tử Vector∗(const Ma trận &, const Vector &);**

**};**



**572** **Nhà điều hành đặc biệt** **Chương 19**

**ma trận lớp {**

**Vectơ v [rc\_max];**

* ***...***

**bạn bè toán tử Vector∗(const Ma trận &, const Vector &);**

**};**

**Bây giờ nhà điều hành∗() có thể tiếp cận với việc thực hiện cả hai Véc tơ và** **Điều đó sẽ cho phép**

**Ma trận. kỹ thuật thực hiện phức tạp, nhưng thực hiện đơn giản sẽ là:**

**Toán tử vectơ∗(const Ma trận & m, const Vector & v) {**

**Véc tơ r;**

**for (int i = 0; i! = rc\_m**

**rv [i] = 0;**

**for (int j = 0; j! = rc\_max; j ++)**

**rv [i] + = mv [i] .v [j] ∗ vv [j];**

**}**

**trả lại r;**

**}**

**MỘT bạn bè khai báo có thể được đặt trong phần riêng tư hoặc phần công khai của một khai báo lớp; nó không quan trọng ở đâu. Giống như một hàm thành viên, một hàm friend được khai báo rõ ràng trong phần khai báo của lớp mà nó là bạn. Do đó, nó là một phần của giao diện đó cũng như một chức năng thành viên.**

**Một hàm thành viên của một lớp có thể là bạn của lớp khác. Ví dụ:**

**class List\_iterator {**

* ***...***

**NS∗ Kế tiếp();**

**};**



**danh sách lớp học {**

**bạn int∗ List\_itera //**

***...***

**};**

**Có một cách viết tắt để biến tất cả các chức năng của lớp này trở thành bạn của lớp khác. Ví dụ:**

**danh sách lớp học {**

**bạn bè lớp**

**List\_iterator; //*...***

**};**

**Cái này bạn bè tuyên bố làm cho tất cả List\_iteratorthành viên của chức năng bạn bè của Danh sách. Khai báo một lớp abạn bè cấp quyền truy cập vào mọi chức năng của lớp đó. Điều đó ngụ ý rằng chúng ta không thể biết tập**

**hợp các hàm có thể truy cập biểu diễn của lớp cấp phép chỉ bằng cách nhìn vào chính lớp đó. Trong trường hợp này, khai báo lớp bạn khác với khai báo hàm thành viên và hàm bạn bè. Rõ ràng, các lớp học bạn bè nên được sử dụng một cách thận trọng và chỉ để thể hiện các khái niệm có mối liên hệ chặt chẽ với nhau.**

**Có thể tạo đối số mẫu là bạn bè:**



**Mục 19.4** **Bạn** **573**

**mẫu <typename T>**

**lớp X {**

**bạn T;**

**bạn cùng lớp T; //*thừa '' lớp ''*// *...***

**};**

**Thông thường, có sự lựa chọn giữa việc đặt một lớp thành thành viên (một lớp lồng nhau) hoặc một người bạn không phải là thành viên (§18.3.1).**

**19.4.1 Tìm bạn**

**Một người bạn trước đây phải là**

**bao quanh một cách ẩn lớp đang khai báo nó là một bạn bè. Các phạm vi bên ngoài phạm vi không gian tên bao quanh trong cùng không được coi là tên được khai báo đầu tiên làbạn bè (§Iso.7.3.1.2). Hãy xem xét một ví dụ kỹ thuật:**

|  |  |  |  |
| --- | --- | --- | --- |
| **lớp C1 {};** | **// *sẽ trở thành bạn của N :: C*** | |  |
| **void f1 ();** | **// *sẽ trở thành bạn của N :: C*** | |  |
| **không gian tên N {** | |  |  |
| **lớp C2 {};** | | **// *sẽ trở thành bạn của C*//** |  |
| **void f2 () {}** | | ***sẽ trở thành bạn của C*** |  |
| **lớp C {** | |  |  |
|  | **int x;** |  |  |
| **công cộng:** | |  |  |
|  | **bạn lớp C1;** | **// *OK (đã xác định trước đó)*** |  |
|  | **bạn bè void f1 ();** |  |  |
|  | **bạn lớp C3** |  |  |
|  | **bạn bè void f3 ();** |  |  |
|  | **bạn lớp C4** |  |  |
| **};** | **bạn bè void f4 ();** |  |  |
|  |  |  |
| **lớp C3 {};** | | **// *bạn của C*** |  |
| **void f3 () {C x; xx = 1; }} // // *OK: bạn của C*** | | |  |
| ***không gian tên N*** | |  |  |
| **lớp C4 {};** |  | **// *không phải bạn của N :: C*// *error: x là riêng tư và f4 ()*** |  |
| **void f4 () {N :: C x; xx = 1; }** | | ***không phải là bạn của N :: C*** |  |



**Một hàm friend có thể được tìm thấy thông qua các đối số của nó (§14.2.4) ngay cả khi nó không được khai báo trong phạm vi kèm theo ngay lập tức. Ví dụ:**

**void f (Ma trận & m)**

**{**

**đảo ngược (m);** **// *Bạn của Matrix invert ()***

**}**



**574** **Nhà điều hành đặc biệt** **Chương 19**

**Do đó, một hàm friend nên được khai báo rõ ràng trong một phạm vi bao quanh hoặc lấy một đối số của lớp của nó hoặc một lớp có nguồn gốc từ đó. Nếu không, người bạn không thể được gọi. Ví dụ:**

* ***không có f () trong phạm vi này***

**lớp X {**

**bạn void f ();** **// *vô dụng***

**bạn bè void h (const X &);** **// *có thể được tìm thấy thông qua lập luận của nó***

**};**

**void g (const X & x) {**

**NS();** **// *không có f () trong s***

**h (x);** **// *Bạn của X h ()***

**}**

**19.4.2 Bạn bè và thành viên**

**Khi nào chúng ta nên sử dụng hàm kết bạn và khi nào thì hàm thành viên là lựa chọn tốt hơn để chỉ định một hoạt động? Đầu tiên, chúng tôi cố gắng giảm thiểu số lượng hàm truy cập vào biểu diễn của một lớp và cố gắng làm cho tập hợp các hàm truy cập phù hợp nhất có thể. Do đó, câu hỏi đầu tiên không phải là '' Có nên là thành viên,tĩnh thành viên, hay một người bạn? " Một số hoạt động phải là thành viên - ví dụ, hàm tạo, hàm hủy và hàm ảo (§3.2.3, §17.2.5) - nhưng thường có một sự lựa chọn. Bởi vì tên thành viên là cục bộ của lớp, một hàm yêu cầu quyền truy cập trực tiếp vào biểu diễn phải là một thành viên trừ khi có lý do cụ thể khiến nó không phải là một thành viên.**

**Xem xét một lớp học NS cung cấp các cách thay thế để trình bày một hoạt động:**

**lớp X {**



* ***...***

**X (int);**

|  |  |  |  |
| --- | --- | --- | --- |
| **int m1** | **();** |  | **// *thành viên*** |
| **int m2** | **() const;** | |  |
| **bạn int f1** | | **(X &);** | **// *bạn bè, không phải thành viên*** |
| **bạn int f2** | | **(const X &);** |  |
| **bạn int f3** | | **(X);** |  |

**};**

**Các hàm thành viên chỉ có thể được gọi cho các đối tượng thuộc lớp của chúng; không có chuyển đổi nào do người dùng xác định được áp dụng cho toán hạng ngoài cùng bên trái của. hoặc -> (nhưng xem §19.2.3). Ví dụ:**

**void g ()**

**{**

**99.m1 ();** **// *lỗi: X (99) .m1 () không được thử***

**99.m2 ();** **// *lỗi: X (99) .m2 () không được thử***

**}**



**Mục 19.4.2** **Bạn bè và thành viên** **575**

**Chức năng toàn cục f1 () có thuộc tính tương tự bởi vì các chuyển đổi ngầm định không được sử dụng cho**

**hăng sô đối số tham chiếu (§7.7).** **Tuy nhiên, các chuyển đổi có thể được áp dụng cho các đối số của f2 ()**

**và f3 ():**

**void h ()**

**{**

|  |  |
| --- | --- |
| **f1 (99);** | **// *lỗi: f1 (X (99)) không được thử: non-const X & đối số*//** |
| **f2 (99);** | ***Được: f2 (X (99)); const X & đối số*// *Được: f3 (X (99));*** |
| **f3 (99);** | ***Đối số X*** |

**}**

**Một hoạt động sửa đổi việc nhập**

**mộthăng sô tài liệu tham khảo**

**Các toán tử sửa đổi một**

**loại do người dùng xác định. Ngược lại, nếu muốn chuyển đổi kiểu ngầm định cho tất cả các toán hạng của một phép toán, thì hàm thực hiện nó phải là một hàm không phải là mộthăng sô đối số tham chiếu hoặc đối số không tham chiếu. Điều này thường xảy ra đối với các toán tử triển khai hàm không yêu cầu toán hạng giá trị khi áp dụng cho các kiểu cơ bản (ví dụ:+, -, và ||). Tuy nhiên, các toán tử như vậy thường cần quyền truy cập vào các biểu diễn của lớp toán hạng của chúng. Do đó, các toán tử nhị phân là nguồn phổ biến nhất của các hàm bạn bè.**

**Trừ khi các chuyển đổi kiểu được xác định, dường như không có lý do thuyết phục nào để chọn một thành viên thay vì một người bạn tham gia đối số hoặc ngược lại. Trong một số trường hợp, người lập trình có thể thích một cú pháp gọi hơn cú pháp khác. Ví dụ: hầu hết mọi người dường như thích ký hiệum2 = inv (m) để tạo ra một sự đảo ngược Ma trận từ NS thay thế m2 = m.inv (). Mặt khác, nếu inv () đảo ngược NS chính nó, thay vì tạo ra một Ma trận đó là nghịch đảo của NS, nó phải là một thành viên.**

**Tất cả những thứ khác được coi là bình đẳng, hãy triển khai các hoạt động cần truy cập trực tiếp vào một biểu diễn dưới dạng các hàm thành viên:**

* **Không thể biết một ngày nào đó ai đó sẽ định nghĩa toán tử chuyển đổi.**
* **Thành viên chức năng**



**một đối số tham khảo**

* **Các biểu thức trong phần thân của một thành viên có thể ngắn hơn đáng kể so với các biểu thức tương đương trong một hàm toàn cục; một hàm nonmember phải sử dụng một đối số rõ ràng, trong khi thành viên có thể sử dụngcái này ngầm hiểu.**
* **Tên thành viên là cục bộ của một lớp, vì vậy chúng có xu hướng ngắn hơn tên của các hàm không phải là phần tử.**
* **Nếu chúng tôi đã xác định một thành viên NS() và sau đó chúng tôi cảm thấy cần phải có một f (x), chúng ta có thể đơn giản định nghĩa nó có nghĩa là xf ().**

**Ngược lại, các phép toán không cần truy cập trực tiếp vào một biểu diễn thường được biểu diễn tốt nhất dưới dạng các hàm không phải là phần tử, có thể trong một không gian tên làm cho mối quan hệ của chúng với lớp rõ ràng (§18.3.6).**



**576** **Nhà điều hành đặc biệt** **Chương 19**

**19.5 Lời khuyên**

1. **Sử dụng nhà điều hành[]() để đăng ký và lựa chọn dựa trên một giá trị duy nhất; §19.2.1.**
2. **Sử dụng nhà điều hành()() cho ngữ nghĩa cuộc gọi, cho chỉ số con và để lựa chọn dựa trên nhiều giá trị; §19.2.2.**
3. **Sử dụng toán tử -> () to dereference '' con trỏ thông minh ''; §19.2.3.**
4. **Tiền tố ưu tiên ++ hơn hậu tố ++; §19.2.4.**
5. **Xác định toàn cầu toán tử mới () và toán tử xóa () chỉ khi bạn thực sự phải làm vậy; §19.2.5.**
6. **Xác định thành viên toán tử mới () và thành viên toán tử xóa () để kiểm soát việc phân bổ và phân bổ các đối tượng của một lớp cụ thể hoặc hệ thống phân cấp của các lớp; §19.2.5.**
7. **Sử dụng lít do người dùng xác định**
8. **Đặt các toán tử theo nghĩa đen**
9. **Đối với các mục đích sử dụng không chuyên biệt, ưu tiên tiêu chuẩn dây (Chương 36) kết quả của các bài tập của riêng**

**bạn; §19.3.**

1. **Sử dụng hàm friend nếu bạn cần một hàm nonmember để có quyền truy cập vào biểu diễn của một lớp (ví dụ: để cải thiện ký hiệu hoặc để truy cập biểu diễn của hai lớp); §19.4.**
2. **Ưu tiên các chức năng thành viên hơn các chức năng bạn bè để cấp quyền truy cập vào việc triển khai một lớp; §19.4.2.**

