



# NGÔN NGỮ LẬP TRÌNH

## Bài 5: Nạp chồng toán tử

Giảng viên: Lý Anh Tuấn

Email: [tuanla@tlu.edu.vn](mailto:tuanla@tlu.edu.vn)

# Nội dung

1. Nạp chồng toán tử cơ sở
  - Các toán tử một ngôi
  - Là hàm thành viên
2. Kiểu đối tượng trả về
3. Hàm bạn, lớp bạn
4. Tham chiếu và nạp chồng
  - << và >>
  - Các toán tử: =, [ ], ++, --

# Giới thiệu nạp chồng toán tử

- Các toán tử  $+$ ,  $-$ ,  $\%$ ,  $==$ , ... thực ra là các hàm
- Chỉ đơn giản được gọi với cú pháp khác:  
 $x + 7$ 
  - “ $+$ ” là toán tử hai ngôi
  - $x$  &  $7$  là các toán hạng
- Hãy tưởng tượng nó là:  
 $+(x, 7)$ 
  - “ $+$ ” là tên hàm
  - $x, 7$  là các đối số
  - Hàm “ $+$ ” trả về tổng của các đối số

# Viễn cảnh nạp chồng toán tử

- Các toán tử dựng sẵn
  - Vd: +, -, =, %, ==, /, \*
  - Đã làm việc với các kiểu C++ dựng sẵn
  - Ở dạng hai ngôi chuẩn
- Chúng ta có thể nạp chồng chúng
  - Để làm việc với các kiểu của chúng ta
  - Để cộng các kiểu theo nhu cầu ở dạng ký hiệu mà chúng ta quen thuộc
- Luôn luôn nạp chồng cho các thao tác tương đương

# Nạp chồng cơ sở

- Nạp chồng toán tử
  - Rất giống nạp chồng hàm
  - Bản thân toán tử là tên của hàm

- Ví dụ khai báo:

```
const Money operator +( const Money& amount1,  
                        const Money& amount2);
```

- Nạp chồng + cho các toán hạng kiểu Money
- Để hiệu quả cần sử dụng các tham chiếu hằng
- Trả về giá trị kiểu Money: cho phép cộng các đối tượng “Money”

# Nạp chồng “+”

- Xét ví dụ trước:
  - Lưu ý: “+” được nạp chồng không phải hàm thành viên
  - Định nghĩa bao gồm nhiều thứ hơn là phép cộng đơn giản
    - Đòi hỏi phát biểu phép cộng kiểu Money
    - Phải điều khiển các giá trị âm/dương
- Các định nghĩa nạp chồng toán tử thường rất đơn giản
  - Chỉ thực hiện “phép cộng” đặc thù cho kiểu của bạn

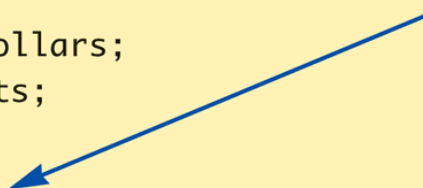

# Định nghĩa “+” Money

- Định nghĩa phép “+” cho lớp Money

```
52  const Money operator +(const Money& amount1, const Money& amount2)
53  {
54      int allCents1 = amount1.getCents( ) + amount1.getDollars( )*100;
55      int allCents2 = amount2.getCents( ) + amount2.getDollars( )*100;
56      int sumAllCents = allCents1 + allCents2;
57      int absAllCents = abs(sumAllCents); //Money can be negative.
58      int finalDollars = absAllCents/100;
59      int finalCents = absAllCents%100;

60      if (sumAllCents < 0)
61      {
62          finalDollars = -finalDollars;
63          finalCents = -finalCents;
64      }

65      return Money(finalDollars, finalCents);
66  }
```



*If the return statements puzzle you, see the tip entitled **A Constructor Can Return an Object.***

# Nạp chồng “==”

- Toán tử đẳng thức, ==
  - Cho phép so sánh các đối tượng Money
  - Khai báo:  
bool operator ==(const Money& amount1,  
const Money& amount2);
    - Trả về kiểu bool với đẳng thức đúng/sai
- Cũng không phải hàm thành viên



# Nạp chồng “==” cho Money

- Định nghĩa toán tử “==” cho lớp Money:

```
83 bool operator ==(const Money& amount1, const Money& amount2)
84 {
85     return ((amount1.getDollars( ) == amount2.getDollars( ))
86             && (amount1.getCents( ) == amount2.getCents( )));
87 }
```

# Kiểu đối tượng trả về

- Trả về đối tượng hằng
  - Việc nạp chồng toán tử “+”  
`const Money operator +(const Money& amount1,  
const Money& amount2);`
  - Trả về một đối tượng hằng
- Trả về đối tượng không hằng
  - Khi không có `const` trong khai báo:  
`Money operator +(const Money& amount1,  
const Money& amount2);`
  - Xét biểu thức được gọi: `m1 + m2`
    - Trả về đối tượng `Money` có thể sửa đổi
- Nên định nghĩa đối tượng trả về là hằng

# Nạp chồng toán tử một ngôi

- C++ có các toán tử một ngôi
  - Được định nghĩa cho một toán hạng
  - Ví dụ, - (phủ định)
    - $x = -y$  // Gán x bằng phủ định của y
  - Các toán tử một ngôi khác:
    - ++, --
- Các toán tử một ngôi cũng có thể được nạp chồng

# Nạp chồng “-” cho Money

- Khai báo hàm nạp chồng “-”
  - Đặt bên ngoài định nghĩa lớp:  
`const Money operator –(const Money& amount);`
  - Lưu ý: chỉ một đối số (vì chỉ có một toán hạng)
- Toán tử “-” được nạp chồng hai lần
  - Với hai toán hạng/đối số (hai ngôi)
  - Với một toán hạng/đối số (một ngôi)
  - Cần có định nghĩa cho cả hai

# Định nghĩa “-” nạp chồng

- Định nghĩa nạp chồng hàm “-”:  

```
const Money operator –(const Money& amount)
{
    return Money(-amount.getDollars(),
                  -amount.getCents());
}
```
- Áp dụng toán tử một ngôi “-” cho kiểu dựng sẵn
  - Là thao tác đã biết đối với các kiểu dựng sẵn

# Sử dụng “-” nạp chồng

- Xét:

```
Money    amount1(10),  
          amount2(6),  
          amount3;  
amount3 = amount1 – amount2;
```

- Gọi nạp chồng “-” hai ngôi

```
amount3.output();    //Displays $4.00  
amount3 = -amount1;
```

- Gọi nạp chồng “-” một ngôi

```
amount3.output();    //Displays -$10.00
```

# Nạp chồng như hàm thành viên

- Trong các ví dụ trước: các hàm là độc lập
  - Được định nghĩa bên ngoài lớp
- Có thể nạp chồng như là “toán tử thành viên”
  - Giống như các hàm thành viên khác
- Khi toán tử là hàm thành viên
  - Chỉ có duy nhất một tham số
  - Đối tượng gọi phục vụ như là tham số thứ nhất

# Ví dụ toán tử thành viên

- Money cost(1, 50), tax(0, 15), total;  
total = cost + tax;
  - Nếu “+” được nạp chồng như là toán tử thành viên:
    - cost là đối tượng gọi
    - tax là đối số duy nhất
  - Hãy hình dung là: total = cost.+(tax);
- Khai báo “+” trong định nghĩa lớp:
  - const Money operator +(const Money& amount);
  - Lưu ý chỉ có một đối số



# Nạp chồng áp dụng hàm ()

- Toán tử gọi hàm, ()
  - Phải được nạp chồng như hàm thành viên
  - Cho phép sử dụng đối tượng lớp giống như một hàm
  - Có thể nạp chồng với số lượng đối số bất kỳ

- Ví dụ:

```
Aclass anObject;  
anObject(42);
```

- Nếu () được nạp chồng → nạp chồng lời gọi

# Các nạp chồng khác

- &&, ||, và toán tử dấu phẩy
  - Phiên bản định nghĩa trước làm việc với kiểu bool
  - Sử dụng đánh giá tắt
  - Khi nạp chồng không sử dụng đánh giá tắt nữa
- Nói chung không nên nạp chồng những toán tử này

# Hàm bạn

- Hàm không phải hàm thành viên
  - Nhắc lại: toán tử nạp chồng là hàm không phải hàm thành viên
    - Chúng truy cập dữ liệu thông qua hàm truy cập và hàm biến đổi
    - Rất kém hiệu quả (phụ phí lời gọi)
- Hàm bạn có thể truy cập trực tiếp dữ liệu lớp private
  - Không phụ phí, hiệu quả hơn
- Do vậy: Tốt nhất là nạp chồng như hàm bạn cho toán tử không phải hàm thành viên

# Hàm bạn

- Hàm bạn của một lớp
  - Không phải hàm thành viên
  - Truy cập trực tiếp tới các thành viên private
    - Giống như cách hàm thành viên làm
- Sử dụng từ khóa *friend* trước khai báo hàm
  - Được đặc tả trong định nghĩa lớp
  - Nhưng không phải là hàm thành viên
- Sử dụng hàm bạn để nạp chồng toán tử
  - Cải thiện hiệu quả thực hiện
  - Tránh gọi hàm thành viên truy cập/biến đổi

# Lớp bạn

- Toàn bộ lớp có thể là bạn
  - Tương tự như hàm là bạn của lớp
  - Ví dụ
    - lớp F là bạn của lớp C
      - Tất cả hàm thành viên lớp F là bạn của C
      - Chiều ngược lại không đúng
- Cú pháp: friend class F
  - Nằm bên trong định nghĩa của lớp cho phép

# Tham chiếu

- Tham chiếu định nghĩa:
  - Tên của một vị trí lưu trữ
  - Tương tự như “con trỏ”
- Ví dụ về tham chiếu đứng độc lập
  - `int robert;`  
`int& bob = robert;`
    - *bob* là tham chiếu tới vị trí lưu trữ của *robert*
    - Những thay đổi với *bob* sẽ ảnh hưởng tới *robert*

# Sử dụng tham chiếu

- Dường như nguy hiểm
- Hữu ích trong một số trường hợp:
- Truyền tham chiếu
  - Thường được sử dụng để thi hành kỹ thuật này
- Trả về một tham chiếu
  - Cho phép các thi hành nạp chồng toán tử được viết tự nhiên hơn
  - Tưởng tượng như là trả về một bí danh cho biến

# Trả về tham chiếu

- Cú pháp:  
`double& sampleFunction(double& variable);`
  - `double&` và `double` là khác nhau
  - Phải giống nhau trong khai báo hàm và đầu đề
- Mục trả về phải có một tham chiếu
  - Chẳng hạn như một biến kiểu đó
  - Không thể là biểu thức chẳng hạn như “`x+5`”
    - Không có vị trí bộ nhớ để trỏ đến
- Ví dụ định nghĩa hàm  
`double& sampleFunction(double& variable)`  
`{`  
    `return variable;`  
`}`



# Nạp chồng << và >>

- Cho phép nhập và xuất các đối tượng
  - Tương tự như nạp chồng các toán tử khác
- Cải thiện tính khả đọc
  - Giống như tất cả các nạp chồng toán tử
  - Cho phép:  
`cout << myObject;`  
`cin >> myObject;`
  - Thay cho:  
`myObject.output(); ...`

# Nạp chồng <<

- Toán tử chèn, <<
  - Sử dụng với cout
  - Là toán tử hai ngôi
- Ví dụ  
cout << "Hello";
  - Toán tử là <<
  - Số hạng thứ nhất là đối tượng cout được định nghĩa trước trong thư viện iostream
  - Số hạng thứ hai là xâu ký tự "Hello"

# Nạp chồng <<

- Các toán hạng của <<
  - Đối tượng cout, thuộc kiểu lớp ostream
  - Kiểu lớp của chúng ta
- Nhắc lại lớp Money
  - Sử dụng hàm thành viên output()
  - Sẽ đẹp hơn nếu có thể sử dụng toán tử <<:  
Money amount(100);  
cout << "I have " << amount << endl;  
thay cho:  
cout << "I have ";  
amount.output()

# Nạp chồng <<

- Money amount(100);  
cout << amount;
  - << nên trả về giá trị nào đó
  - Cho phép lồng nhau:  
cout << "I have " << amount;  
(cout << "I have ") << amount;
  - Đối tượng cout
    - Trả về kiểu đối số thứ nhất của nó, ostream

# Ví dụ nạp chồng << và >>

## Display 8.5 Overloading << and >>

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <cmath>
4  using namespace std;

5  //Class for amounts of money in U.S. currency
6  class Money
7  {
8  public:
9      Money( );
10     Money(double amount);
11     Money(int theDollars, int theCents);
12     Money(int theDollars);
13     double getAmount( ) const;
14     int getDollars( ) const;
15     int getCents( ) const;
16     friend const Money operator +(const Money& amount1, const Money& amount2)
17     friend const Money operator -(const Money& amount1, const Money& amount2)
18     friend bool operator ==(const Money& amount1, const Money& amount2);
19     friend const Money operator -(const Money& amount);
20     friend ostream& operator <<(ostream& outputStream, const Money& amount);
21     friend istream& operator >>(istream& inputStream, Money& amount);
22 private:
23     int dollars; //A negative amount is represented as negative dollars and
24     int cents; //negative cents. Negative $4.50 is represented as -4 and -50.
```

# Ví dụ nạp chồng << và >>

```
25     int dollarsPart(double amount) const;
26     int centsPart(double amount) const;
27     int round(double number) const;
28 };

29 int main( )
30 {
31     Money yourAmount, myAmount(10, 9);
32     cout << "Enter an amount of money: ";
33     cin >> yourAmount;
34     cout << "Your amount is " << yourAmount << endl;
35     cout << "My amount is " << myAmount << endl;
36
37     if (yourAmount == myAmount)
38         cout << "We have the same amounts.\n";
39     else
40         cout << "One of us is richer.\n";

41     Money ourAmount = yourAmount + myAmount;
```

# Ví dụ nạp chồng << và >>

## Display 8.5 Overloading << and >>

```
42     cout << yourAmount << " + " << myAmount
43         << " equals " << ourAmount << endl;

44     Money diffAmount = yourAmount - myAmount;
45     cout << yourAmount << " - " << myAmount
46         << " equals " << diffAmount << endl;

47     return 0;
48 }
```

*Since << returns a reference, you can chain << like this. You can chain >> in a similar way.*

*<Definitions of other member functions are as in Display 8.1. Definitions of other overloaded operators are as in Display 8.3.>*

```
49 ostream& operator <<(ostream& outputStream, const Money& amount)
50 {
51     int absDollars = abs(amount.dollars);
52     int absCents = abs(amount.cents);
53     if (amount.dollars < 0 || amount.cents < 0)
54         //accounts for dollars == 0 or cents == 0
55         outputStream << "$-";
56     else
57         outputStream << '$';
58     outputStream << absDollars;
```

*In the main function, cout is plugged in for outputStream.*

*For an alternate input algorithm, see Self-Test Exercise 3 in Chapter 7.*

# Ví dụ nạp chồng << và >>

```
59     if (absCents >= 10)
60         outputStream << '.' << absCents;
61     else
62         outputStream << '.' << '0' << absCents;

63     return outputStream;
64 }
65
66 //Uses iostream and cstdlib:
67 istream& operator >>(istream& inputStream, Money& amount)
68 {
69     char dollarSign;
70     inputStream >> dollarSign; //hopefully
71     if (dollarSign != '$')
72     {
73         cout << "No dollar sign in Money input.\n";
74         exit(1);
75     }

76     double amountAsDouble;
77     inputStream >> amountAsDouble;
78     amount.dollars = amount.dollarsPart(amountAsDouble);
```

*Returns a reference*

*In the main function, cin is plugged in for inputStream.*

*Since this is not a member operator, you need to specify a calling object for member functions of Money.*

(continued)



# Ví dụ nạp chồng << và >>

## Display 8.5 Overloading << and >>

```
79 amount.cents = amount.centsPart(amountAsDouble);  
80 return inputStream;  
81 }
```

*Returns a reference*

### SAMPLE DIALOGUE

Enter an amount of money: \$123.45  
Your amount is \$123.45  
My amount is \$10.09.  
One of us is richer.  
\$123.45 + \$10.09 equals \$133.54  
\$123.45 - \$10.09 equals \$113.36

# Toán tử gán, =

- Phải được nạp chồng như toán tử thành viên
- Được nạp chồng tự động
  - Toán tử gán mặc định
    - Sao chép thông minh thành viên
    - Các biến thành viên từ một đối tượng  $\rightarrow$  tương ứng với các biến thành viên từ một đối tượng khác
- Mặc định cho các lớp đơn giản
  - Nhưng với các con trỏ  $\rightarrow$  phải tự viết

# Tăng và giảm

- Mỗi toán tử có hai phiên bản
  - Ký hiệu tiền tố:  $++x$ ;
  - Ký hiệu hậu tố:  $x++$ ;
- Phải phân biệt khi nạp chồng
  - Phương pháp nạp chồng chuẩn  $\rightarrow$  tiền tố
  - Thêm tham số thứ hai kiểu `int`  $\rightarrow$  hậu tố
    - Chỉ để đánh dấu giúp bộ biên dịch
    - Chỉ rõ hậu tố được cho phép

# Nạp chồng toán tử mảng, [ ]

- Có thể nạp chồng [ ] cho lớp
  - Để sử dụng với các đối tượng thuộc lớp
  - Toán tử phải trả về một tham chiếu
  - Toán tử [ ] phải là một hàm thành viên

# Tóm tắt

- Các toán tử C++ dựng sẵn có thể được nạp chồng
- Các toán tử thực ra chỉ là các hàm
- Hàm bạn truy cập trực tiếp thành viên private
- Các toán tử có thể được nạp chồng như là các hàm thành viên
  - Số hạng thứ nhất là đối tượng gọi
- Hàm bạn giúp tăng hiệu quả
  - Không cần thiết nếu đã có các hàm truy cập/hàm biến đổi
- Tham chiếu đặt tên cho biến bằng một bí danh
- Có thể nạp chồng <<, >>
  - Kiểu trả về là một tham chiếu