

# Bài 8: Nạp chồng toán tử, Từ khóa friend và Tham chiếu

Giảng viên: Hoàng Thị Điệp

Khoa Công nghệ Thông tin – ĐH Công Nghệ

# ABSOLUTE C++

ANSI/ISO STANDARD

STANDARD TEMPLATE  
LIBRARY

TEMPLATES

NAMESPACES

STRINGS

VECTORS

VIRTUAL FUNCTIONS

EXCEPTION HANDLING

STREAM I/O

UML

ENCAPSULATION

PATTERNS

**4**<sup>TH</sup>  
EDITION

SAVITCH

## Chapter 8

### Operator Overloading, Friends, and References

# Mục tiêu bài học

- Căn bản về nạp chồng toán tử
  - Toán tử một ngôi
  - Nạp chồng dưới dạng hàm thành viên
- Từ khóa friend và chuyển đổi kiểu tự động
  - Hàm friend, lớp friend
  - Hàm kiến tạo và chuyển đổi kiểu tự động
- Tham chiếu và bàn thêm về nạp chồng
  - << và >>
  - Các toán tử: = , [], ++, --

# Giới thiệu về nạp chồng toán tử

- Các toán tử  $+$ ,  $-$ ,  $\%$ ,  $==$ , v.v.
  - thực ra là các hàm!
- Đơn giản là chúng được “gọi” bằng cú pháp khác:  
 $x + 7$ 
  - “+” là toán tử 2 ngôi với toán hạng là  $x$  và  $7$
  - Con người “thích” kí hiệu này hơn
- Hãy nghĩ về nó như là:  
 $+(x, 7)$ 
  - “+” là tên hàm
  - $x, 7$  là đối số
  - Hàm “+” trả về “tổng” của các đối số của nó

# Nạp chồng toán tử

- Các toán tử có sẵn
  - Ví dụ: `+`, `-`, `=`, `%`, `==`, `/`, `*`
  - Đã làm việc với các kiểu có sẵn của C++
  - Với kí hiệu “hai ngôi” chuẩn
- Ta có thể nạp chồng chúng
  - Để làm việc với kiểu dữ liệu của ta!
  - Để cộng “các biến Chair” hoặc “các biến Money”
    - Phù hợp với nhu cầu của ta
    - Theo “kí hiệu” mà ta quen dùng
- Hãy nạp chồng bằng “công việc” tương tự!

# Căn bản về nạp chồng

- Nạp chồng toán tử
  - Có nhiều điểm tương tự với nạp chồng hàm
  - Toán tử chính là tên của hàm
- Ví dụ: khai báo

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

  - Nạp chồng phép + cho 2 toán hạng kiểu Money
  - Sử dụng tham số tham chiếu hằng cho hiệu quả
  - Giá trị trả về có kiểu Money
    - Cho phép cộng các đối tượng "Money"

# Phép “+” đã được nạp chồng

- Từ ví dụ vừa rồi:
  - Lưu ý: phép cộng nạp chồng không phải là hàm thành viên
  - Định nghĩa trong Display 8.1 làm nhiều việc hơn phép cộng đơn thuần
    - Phần bắt buộc: xử lý việc cộng 2 đối tượng Money
    - Xử lý các giá trị âm/dương
- Định nghĩa nạp chồng toán tử nhìn chung là đơn giản
  - Chỉ cần thực hiện “phép cộng” cho kiểu “của bạn”

# Định nghĩa “+” cho Money

## Display 8.1 Nạp chồng toán tử

- 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.***



# Phép “==” nạp chồng

- Phép so sánh bằng, ==
  - Cho phép so sánh các đối tượng Money
  - Khai báo:

```
bool operator ==( const Money& amount1,  
                  const Money& amount2);
```

    - Trả về kiểu bool cho đẳng thức đúng/sai
  - Đây cũng không phải là một hàm thành viên (giống như phép “+” nạp chồng)

# Phép “==” nạp chồng cho lớp Money:

## Display 8.1 Nạp chồng toán tử

- Định nghĩa phép “==” 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 }
```

# Hàm kiến tạo trả về đối tượng

- Hàm kiến tạo là hàm kiểu void?
  - Ta “nghĩ” như vậy nhưng không phải
  - Nó là một hàm “đặc biệt”
    - Với các tính chất đặc biệt
    - CÓ THỂ trả về một giá trị!
- Nhắc lại: câu lệnh `return` trong phép “+” nạp chồng của kiểu `Money`:
  - `return Money(finalDollars, finalCents);`
    - Trả về một “lời gọi” tới lớp `Money`!
    - Suy ra hàm kiến tạo thực ra “có trả về” một đối tượng!
    - Gọi là “đối tượng vô danh”

# Trả về giá trị const

- Ta lại xét việc nạp chồng phép “+”:  
`const Money operator +(const Money& amount1,  
const Money& amount2);`
  - Trả về một “đối tượng hằng”?
  - Vì sao?
- Để hiểu được lý do, hãy xét ảnh hưởng của việc trả về đối tượng không chỉ định là `const...` →

# Trả về một giá trị không chỉ định là const

- Hãy xem xét khai báo không chỉ định trả về const:  
`Money operator +(const Money& amount1,  
const Money& amount2);`
- Xét biểu thức gọi tới nó:  
`m1 + m2`
  - Trong đó m1 & m2 là các đối tượng Money
  - Đối tượng trả về có kiểu Money
  - Ta có thể “thao tác trên” đối tượng trả về!
    - Ví dụ như gọi tới hàm thành viên...

# Thao tác trên đối tượng không chỉ định const

- Có thể gọi các hàm thành viên:
  - Ta có thể gọi tới hàm thành viên trên đối tượng trả về bởi biểu thức `m1+m2`:
    - `(m1+m2).output();` // Hợp lệ?
      - Đây không phải là vấn đề vì nó chẳng làm biến đổi giá trị vừa trả về
    - `(m1+m2).input();` // Hợp lệ!
      - Đây là vấn đề! Hợp lệ nhưng biến đổi đối tượng vừa trả về!
    - Có thể biến đổi đối tượng “vô danh”!
    - Không cho phép điều đó ở đây!
- Vì vậy ta định nghĩa đối tượng trả về là `const`

# Nạp chồng toán tử một ngôi

- C++ có toán tử một ngôi:
  - Là toán tử nhận một toán hạng
  - Ví dụ: phép phủ định -
    - `x = -y;` // Đặt x bằng âm y
  - Toán tử một ngôi khác:
    - `++, --`
- Toán tử một ngôi cũng có thể bị nạp chồng

# Nạp chồng phép phủ định “-” cho kiểu Money

- Khai báo hàm “-” nạp chồng
  - Đặt ngoài định nghĩa lớp:  
`const Money operator –(const Money& amount);`
  - Chú ý là chỉ có một đối số
    - Vì phép toán một ngôi chỉ có 1 toán hạng
- Phép “-” bị nạp chồng 2 lần!
  - Cho 2 toán hạng/đối số (hai ngôi)
  - Cho 1 toán hạng/đối số (một ngôi)
  - Phải cung cấp cả 2 định nghĩa



# Định nghĩa nạp chồng “-”

- Định nghĩa hàm “-” nạp chồng  
`const Money operator –(const Money& amount)`  
{  
    return Money( -amount.getDollars(),  
                  -amount.getCents());  
}
- Áp dụng toán tử một ngôi “-” cho kiểu có sẵn
  - Phép toán này là “đã biết” với những kiểu có sẵn
- Trả về đối tượng vô danh

# Sử dụng phép “-” nạp chồng

- Xét:

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

```
amount3 = amount1 - amount2;
```

- Sẽ gọi tới phép “-” nạp chồng hai ngôi

```
amount3.output();    // Hiển thị $4.00
```

```
amount3 = -amount1;
```

- Sẽ gọi tới phép “-” nạp chồng một ngôi

```
amount3.output();    // Hiển thị -$10.00
```

# Nạp chồng dưới dạng hàm thành viên

- Ví dụ trước: các hàm nạp chồng toán tử đứng độc lập
  - Định nghĩa bên ngoài lớp
- Có thể nạp chồng dưới dạng “toán tử thành viên”
  - Được xem như “hàm thành viên” như các hàm khác
- Khi toán tử là hàm thành viên:
  - Toán tử hai ngôi chỉ cần một tham số, không phải 2!
  - Đối tượng gọi giữ vai trò tham số thứ nhất

# Hoạt động của toán tử thành viên

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

# Hàm const

- Khi nào thì chỉ định hàm là `const`?
  - Hàm thành viên `const` không được phép biến đổi dữ liệu thành viên của lớp
  - Đối tượng `const` chỉ có thể gọi tới hàm thành viên `const`
- Phong cách lập trình tốt yêu cầu:
  - Bất cứ hàm thành viên nào không biến đổi dữ liệu thành viên cần được chỉ định là `const`
- Sử dụng từ khóa `const` cuối khai báo hàm và dòng đầu hàm

# Nạp chồng toán tử: Cách nào?

- Lập trình hướng đối tượng
  - Các nguyên lý khuyến khích cài đặt toán tử thành viên
  - Được nhất trí nhiều. Duy trì “tinh thần” của LTHĐT
- Toán tử thành viên thì hiệu quả hơn
  - Không cần gọi tới các hàm truy cập và hàm biến đổi
- Ít nhất một điểm yếu quan trọng
  - (Sẽ bàn sau)

# Nạp chồng ứng dụng hàm ()

- Toán tử gọi hàm, ()
  - Nếu nạp chồng, phải cài đặt dạng hàm thành viên
  - Cho phép dùng đối tượng của lớp như một hàm
  - Có thể nạp chồng cho số lượng đối số bất kì
- Ví dụ:  
`Aclass anObject;`  
`anObject(42);`
  - Nếu nạp chồng () → gọi đến hàm nạp chồng đó

# Các phép nạp chồng khác

- &&, ||, và toán tử dấu phẩy
  - Phiên bản định nghĩa sẵn làm việc với kiểu bool
  - Nhắc lại: nó sử dụng “tính giá trị biểu thức kiểu đoạn mạch” (short-circuit evaluation)
  - Khi nạp chồng, đoạn mạch không còn hiệu lực
    - Thay vào đó “tính giá trị đầy đủ” được sử dụng
    - Trái với mong đợi
- Nhìn chung, không nên nạp chồng các toán tử này



# Hàm friend

- Nạp chồng dưới dạng hàm ngoài lớp (không phải thành viên)
  - Nhắc lại: khi nạp chồng toán tử dưới dạng hàm ngoài lớp
    - Chúng truy cập dữ liệu thông qua các hàm thành viên truy cập và hàm thành viên biến đổi
    - Không hiệu quả (chi phí của các lời gọi)
- Từ khóa friend giúp truy cập trực tiếp vào dữ liệu của lớp
  - Không chi phí, hiệu quả hơn
- Do đó: tốt nhất là nạp chồng toán tử dưới dạng hàm ngoài lớp dùng từ khóa friend!

# Hàm friend

- Hàm **friend** của một lớp
  - Không phải là hàm thành viên
  - Được truy cập trực tiếp tới các thành viên **private**
    - Giống như các hàm thành viên
- Dùng từ khóa **friend** trước khai báo hàm
  - Chỉ định TRONG định nghĩa lớp
  - Nhưng chúng không phải là hàm thành viên!

# Sử dụng hàm friend

- Nạp chồng toán tử
  - Cách dùng thường gặp nhất với friend
  - Cải thiện hiệu quả
  - Tránh phải gọi tới các hàm thành viên truy cập/biến đổi
  - Toán tử cần được truy cập
    - Có thể trao quyền truy cập toàn bộ như hàm friend
- friend có thể là bất cứ hàm nào

# Tính thuần túy của hàm friend

- friend không thuần túy?
  - “Tình thần“ của LTHĐT yêu cầu tất cả toán tử và hàm phải là hàm thành viên
  - Nhiều người nghĩ rằng friend vi phạm nguyên lý cơ bản của LTHĐT
- Có ích?
  - Với toán tử: rất có ích!
  - Cho phép chuyển đổi kiểu tự động
  - Vẫn đóng gói: friend nằm trong định nghĩa lớp
  - Cải thiện hiệu quả

# Lớp friend

- Một lớp có thể là friend của lớp khác
  - Tương tự như việc hàm là friend của lớp
  - Ví dụ:  
lớp F là friend của lớp C
    - Tất cả các hàm thành viên của lớp F là friend của C
    - Không thuận nghịch (tức: chiều ngược lại không đúng)
    - Tình bạn có thể được ban tặng nhưng không thể đòi hỏi
- Cú pháp: **friend class F;**
  - Nằm bên trong định nghĩa của lớp “cấp quyền”

# Tham chiếu

- Tham chiếu là:
  - Tên của nơi lưu trữ
  - Tương tự như “con trỏ”
- Ví dụ về tham chiếu đứng độc lập:
  - `int robert;`
  - `int& bob = robert;`
    - `bob` tham chiếu tới nơi lưu trữ cho `robert`
    - Biến đổi trên `bob` sẽ tác động tới `robert`
- Dễ nhầm lẫn?

# Sử dụng tham chiếu

- Có vẻ nguy hiểm
- Hữu ích trong một số trường hợp:
- Truyền tham chiếu
  - Thường dùng để cài đặt cơ chế này
- Trả về một tham chiếu
  - Cho phép cài đặt toán tử nạp chồng được tự nhiên hơn
  - Hãy nghĩ nó trả về “bí danh” của một biến

# Trả về tham chiếu

- Cú pháp:  
`double& sampleFunction(double& variable);`
  - `double&` khác `double`
  - Khai báo hàm và dòng đầu định nghĩa hàm phải khớp nhau
- Thực thể được trả về phải “có” một tham chiếu
  - Như là một biến cùng kiểu
  - Không thể là một biểu thức, ví dụ “ $x+5$ ”
    - Vì biểu thức không được cấp một nơi nào đó trong bộ nhớ để ta tham chiếu tới



# Trả về tham chiếu trong định nghĩa

- Ví dụ định nghĩa hàm:  
`double& sampleFunction(double& variable)`  
{  
    return variable;  
}
- Ví dụ không quan trọng, vô dụng
- Chỉ để biểu diễn khái niệm
- Ứng dụng chính:
  - Nạp chồng một số toán tử nhất định

# Nạp chồng << và >>

- Cho phép ghi và đọc đối tượng dùng các luồng chuẩn
  - Tương tự như nạp chồng các toán tử khác
  - Có thêm một số điểm tinh vi
- Dễ đọc hơn
  - Như tất cả các toán tử nạp chồng khác
  - Cho phép:  
`cout << myObject;`  
`cin >> myObject;`
  - Thay cho:  
`myObject.output();`

# Nạp chồng <<

- Toán tử chèn, <<
  - Dùng với `cout`
  - Là toán tử hai ngôi
- Ví dụ:  
`cout << "Hello";`
  - Toán tử là <<
  - Toán hạng thứ nhất là đối tượng định nghĩa sẵn `cout`
    - Từ thư viện `iostream`
  - Toán hạng thứ 2 là xâu hằng `"Hello"`

# Nạp chồng <<

- Các toán hạng của <<
  - Đối tượng cout kiểu ostream
  - Đối tượng của lớp ta định nghĩa
- Nhắc lại: lớp Money
  - Dùng hàm thành viên output()
  - Tốt hơn nếu ta có thể dùng toán tử <<:  
`Money amount(100);`  
`cout << "I have " << amount << endl;`  
thay cho:  
`cout << "I have ";`  
`amount.output()`

# Giá trị trả về của << nạp chồng

- Money amount(100);  
cout << amount;
  - << nên trả về giá trị nào đó
  - Để cho phép hiện tượng thác lũ:  
cout << "I have " << amount;  
(cout << "I have ") << amount;
    - 2 lệnh trên là tương đương
- Trả về gì?
  - Chính đối tượng `cout`
    - Trả về kiểu của đối số đầu tiên, `ostream&`

# Ví dụ nạp chồng <<:

## Display 8.5 Nạp chồng << và >> (1/5)

### 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 <<:

## Display 8.5 Nạp chồng << và >> (2/5)

```
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 <<:

## Display 8.5 Nạp chồng << và >> (3/5)

### 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 <<:

## Display 8.5 Nạp chồng << và >> (4/5)

```
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.

# Ví dụ nạp chồng <<: Display 8.5 Nạp chồng << và >> (5/5)

## Display 8.5 Overloading << and >>

---

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

### 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 dưới dạng toán tử thành viên
- Luôn được nạp chồng tự động
  - Toán tử gán mặc định:
    - Sao chép các thành viên
    - Biến thành viên từ một đối tượng  $\rightarrow$  biến thành viên tương ứng của đối tượng khác
- Toán tử gán mặc định là chấp nhận được với những lớp đơn giản
  - Nhưng khi có con trỏ  $\rightarrow$  ta phải nạp chồng toán tử gán!

# Tự tăng và tự giảm

- Mỗi toán tử có 2 phiên bản
  - Kí hiệu trước: `++x`;
  - Kí hiệu sau: `x++`;
- Phải phân biệt khi nạp chồng
  - Cách nạp chồng chuẩn → Kí hiệu trước
  - Thêm một tham số kiểu `int` → Kí hiệu sau
    - Chỉ là đánh dấu cho trình biên dịch!
    - Được phép chỉ định kí hiệu sau

# Nạp chồng toán tử mảng, [ ]

- Có thể nạp chồng [ ] cho lớp của bạn
  - Để dùng với đối tượng của lớp đó
  - Toán tử phải trả về một tham chiếu!
  - Toán tử [ ] phải là hàm thành viên!

# Tóm tắt 1

- Có thể nạp chồng các toán tử có sẵn của C++
  - Để làm việc với các đối tượng của kiểu do người dùng định nghĩa
- Toán tử thật ra là hàm
- Hàm friend có quyền truy cập trực tiếp vào thành viên private
- Toán tử có thể được nạp chồng như hàm thành viên
  - Toán hạng thứ nhất chính là đối tượng gọi tới hàm thành viên này

## Tóm tắt 2

- Hàm friend chỉ làm tăng tính hiệu quả
  - Không bắt buộc nếu có đủ hàm thành viên truy cập/biến đổi
- Tham chiếu “đặt” cho một biến một bí danh nào đó
- Có thể nạp chồng <<, >>
  - Kiểu trả về là tham chiếu tới kiểu stream

# Chuẩn bị bài tới

- Đọc chương 9 giáo trình: Xâu