

Bài 3: Căn bản về hàm

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

4TH
EDITION

SAVITCH

Chapter 3

Function Basics

Mục tiêu bài học

- Các hàm định nghĩa sẵn
 - Hàm có trả về giá trị
 - Hàm không trả về giá trị
- Các hàm không có sẵn
 - Khai báo hàm, Định nghĩa hàm, Gọi hàm
 - Hàm đệ quy
- Các quy tắc về phạm vi hoạt động
 - Biến cục bộ
 - Hằng toàn cục và biến toàn cục
 - Khối, phạm vi lồng nhau

Giới thiệu về hàm

- Là các khối tạo nên chương trình
- Thuật ngữ trong các ngôn ngữ lập trình khác:
 - Thủ tục, chương trình con, phương thức
 - Trong C++: hàm
- I-P-O
 - Input – Process – Output
 - Là các phần cơ bản cấu thành chương trình
 - Dùng hàm cho từng phần này

Các hàm định nghĩa sẵn

- Ta có thể dùng rất nhiều hàm có sẵn trong các thư viện!
- Có hai loại:
 - Những hàm có trả về giá trị
 - Những hàm không trả về giá trị (void)
- Ta phải `#include` thư viện phù hợp
 - Ví dụ:
 - `<cmath>`, `<cstdlib>` (là những thư viện gốc "C")
 - `<iostream>` (để dùng `cout`, `cin`)

Sử dụng hàm định nghĩa sẵn

- Có rất nhiều hàm toán học
 - Có thể thấy trong thư viện `<cmath>`
 - Hầu hết trả về một giá trị (“đáp số”)
- Ví dụ: `theRoot = sqrt(9.0);`
 - Các thành phần:
 - `sqrt` = tên của hàm trong thư viện
 - `theRoot` = biến dùng để ghi đáp số
 - `9.0` = đối số hay “đầu vào” của hàm
 - Theo I-P-O:
 - I = 9.0
 - P = “tính căn bậc hai”
 - O = 3, là giá trị trả về của hàm, sẽ được gán cho `theRoot`

Lời gọi hàm

- Trở lại phép gán:
`theRoot = sqrt(9.0);`
 - Biểu thức "`sqrt(9.0)`" được gọi là lời gọi hàm (function *call* hay function *invocation*)
 - Đối số trong một lời gọi hàm (`9.0`) có thể là một giá trị hằng, một biến hoặc một biểu thức
 - Bản thân lời gọi có thể là một phần của một biểu thức:
 - `bonus = sqrt(sales)/10;`
 - Bất cứ nơi nào là hợp lệ cho kiểu trả về của hàm thì bạn có thể đặt lời gọi hàm.

Ví dụ lớn hơn:

Display 3.1 Một hàm có sẵn có trả về một giá trị (1/2)

Display 3.1 A Predefined Function That Returns a Value

```
1 //Computes the size of a doghouse that can be purchased
2 //given the user's budget.
3 #include <iostream>
4 #include <cmath>
5 using namespace std;

6 int main( )
7 {
8     const double COST_PER_SQ_FT = 10.50;
9     double budget, area, lengthSide;

10    cout << "Enter the amount budgeted for your doghouse $";
11    cin >> budget;

12    area = budget/COST_PER_SQ_FT;
13    lengthSide = sqrt(area);
```


Ví dụ lớn hơn: Display 3.1 Một hàm có sẵn có trả về một giá trị (2/2)

```
14     cout.setf(ios::fixed);
15     cout.setf(ios::showpoint);
16     cout.precision(2);
17         cout << "For a price of $" << budget << endl
18             << "I can build you a luxurious square doghouse\n"
19             << "that is " << lengthSide
20             << " feet on each side.\n";

21     return 0;
22 }
```

SAMPLE DIALOGUE

```
Enter the amount budgeted for your doghouse $25.00
For a price of $25.00
I can build you a luxurious square doghouse
that is 1.54 feet on each side.
```

Các hàm định nghĩa sẵn (tiếp)

- `#include <cstdlib>`

- Thư viện này chứa các hàm:

- `abs()` // Trả về giá trị tuyệt đối của một biến int
- `labs()` // Trả về giá trị tuyệt đối của một biến long int
- `*fabs()` // Trả về giá trị tuyệt đối của một biến float

- `*fabs()` thực ra nằm trong thư viện `<cmath>`!

- Có thể gây bối rối
- Hãy nhớ rằng các thư viện được bổ sung dần dần sau khi C++ “chào đời”
- Hãy tham khảo chi tiết ở các phụ lục/các sách hướng dẫn sử dụng

Các hàm toán học

- `pow(x, y)`
 - Trả về x mũ y
`double result, x = 3.0, y = 2.0;`
`result = pow(x, y);`
`cout << result;`
 - In ra màn hình 9.0 (vì $3.0^{2.0} = 9.0$)
- Chú ý là hàm này nhận hai đối số
 - Số lượng đối số của hàm có thể là con số bất kì. Kiểu của chúng cũng có thể khác nhau.

Nói thêm về hàm toán học:

Display 3.2 Một số hàm định nghĩa sẵn (1/2)

Display 3.2 Some Predefined Functions

NAME	DESCRIPTION	TYPE OF ARGUMENTS	TYPE OF VALUE RETURNED	EXAMPLE	VALUE	LIBRARY HEADER
sqrt	Square root	double	double	sqrt(4.0)	2.0	cmath
pow	Powers	double	double	pow(2.0, 3.0)	8.0	cmath
abs	Absolute value for <code>int</code>	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Absolute value for <code>long</code>	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Absolute value for <code>double</code>	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath

Nói thêm về hàm toán học:

Display 3.2 Một số hàm định nghĩa sẵn (2/2)

<code>ceil</code>	Ceiling (round up)	<code>double</code>	<code>double</code>	<code>ceil(3.2)</code> <code>ceil(3.9)</code>	4.0 4.0	<code>cmath</code>
<code>floor</code>	Floor (round down)	<code>double</code>	<code>double</code>	<code>floor(3.2)</code> <code>floor(3.9)</code>	3.0 3.0	<code>cmath</code>
<code>exit</code>	End pro- gram	<code>int</code>	<code>void</code>	<code>exit(1);</code>	None	<code>cstdlib</code>
<code>rand</code>	Random number	None	<code>int</code>	<code>rand()</code>	Varies	<code>cstdlib</code>
<code>srand</code>	Set seed for rand	<code>unsigned int</code>	<code>void</code>	<code>srand(42);</code>	None	<code>cstdlib</code>

Các hàm void định nghĩa sẵn

- Không có giá trị trả về
- Làm một việc gì đó nhưng không cho bạn một “đáp số”
- Khi được gọi, bản thân nó là một lệnh
 - `exit(1);` // không có giá trị trả về
// do đó không dùng được trong phép gán
 - Lệnh này kết thúc chương trình
 - Các hàm void vẫn có thể có đối số
- Tất cả các đặc điểm đều giống hệt hàm “có trả về một giá trị”
 - Chúng đơn giản không trả về một giá trị mà thôi!

Sinh số ngẫu nhiên

- Trả về một số được chọn ngẫu nhiên
- Dùng để viết chương trình mô phỏng hay games
 - `rand()`
 - Không đổi số
 - Trả về một số trong khoảng 0 đến `RAND_MAX`
 - Scaling: phép vị tự/ co dãn biên độ
 - Ép số ngẫu nhiên này vào khoảng nhỏ hơn
`rand() % 6`
 - Trả về một giá trị ngẫu nhiên giữa 0 & 5
 - Shifting: phép tịnh tiến
`rand() % 6 + 1`
 - Tịnh tiến miền giá trị thành từ 1 đến 6 (có thể mô phỏng kết quả tung xúc xắc)

Nhân của số ngẫu nhiên

- Các số giả ngẫu nhiên
 - Các lời gọi tới hàm `rand()` sinh ra “chuỗi” các số ngẫu nhiên biết trước
- Sử dụng “nhân” (seed) để thay đổi chuỗi này
 - `srand(giá_trị_nhân);`
 - là hàm `void`
 - nhận một đối số, là “nhân”
 - có thể dùng bất cứ giá trị nào làm nhân, ví dụ thời gian hệ thống:
`srand(time(0));`
 - `time()` trả về thời gian hệ thống dưới dạng giá trị số
 - các hàm liên quan thời gian được định nghĩa sẵn trong thư viện `<time>`

Ví dụ sinh số ngẫu nhiên

- Số ngẫu nhiên kiểu double trong khoảng 0.0 và 1.0:
`(RAND_MAX - rand())/static_cast<double>(RAND_MAX)`
 - Ở đây sử dụng phép chuyển đổi kiểu để ép phép chia với độ chính xác `double`
- Số ngẫu nhiên kiểu int trong khoảng 1 và 6:
`rand() % 6 + 1`
 - "%" là phép lấy dư
- Số ngẫu nhiên kiểu int trong khoảng 10 và 20:
`rand() % 10 + 10`

Các hàm không có sẵn

- Bạn cần viết hàm của riêng mình.
- Hàm là những khối tạo nên chương trình
 - Chia để trị
 - Dễ đọc
 - Dễ tái sử dụng
- “Định nghĩa” bạn viết ra có thể đặt:
 - Cùng tệp với hàm `main()`
 - Hoặc ở một tệp riêng biệt để những chương trình khác cũng có thể gọi tới nó

Làm việc với hàm

- 3 khái niệm quan trọng khi làm việc với hàm:
 - Khai báo hàm/ Nguyên mẫu hàm
 - Chứa thông tin cho trình biên dịch
 - Để có thể thông dịch chính xác cho các lời gọi
 - Định nghĩa hàm
 - Là mã/cài đặt thực sự cho thấy hàm làm gì
 - Lời gọi hàm
 - Truyền điều khiển cho hàm

Khai báo hàm

- Còn gọi là nguyên mẫu hàm
- Là khai báo mang thông tin cho trình biên dịch
- Cho trình biên dịch biết cần dịch các lời gọi như thế nào
 - Cú pháp:
`<kiểu_trả_về> tênHàm(<danh-sách-tham-số-hình-thức>);`
 - Ví dụ:
`double totalCost(int numberParameter,
 double priceParameter);`
- Phải được đặt trước tất cả các lời gọi tới hàm đó
 - Bên trong vùng khai báo của `main()`
 - Hoặc ở bên trên `main()` trong vùng toàn cục

Định nghĩa hàm

- Là cài đặt của hàm
- Cũng giống như khi bạn viết hàm `main()`

- Ví dụ:

```
double totalCost(           int numberParameter,
                           double priceParameter)
{
    const double TAXRATE = 0.05;
    double subTotal;
    subtotal = priceParameter * numberParameter;
    return (subtotal + subtotal * TAXRATE);
}
```

- Chú ý lùi đầu dòng hợp lý.

Vị trí đặt định nghĩa hàm

- Đặt phía dưới hàm `main()`
 - KHÔNG PHẢI “bên trong” hàm `main()`!
- Các hàm “bình đẳng” với nhau; không hàm nào là con của hàm nào
- Các tham số hình thức trong định nghĩa
 - Là “chỗ đặt trước” cho dữ liệu truyền vào
 - “Tên biến” được sử dụng để tham chiếu tới các dữ liệu trong định nghĩa
- Lệnh `return`
 - Truyền dữ liệu cho nơi gọi hàm

Lời gọi hàm

- Giống với gọi hàm có sẵn
`bill = totalCost(number, price);`
- Nhắc lại: `totalCost` trả về một giá trị double
 - giá trị này sẽ được gán cho biến tên là "`bill`"
- Đối số ở đây là: `number, price`
 - Đối số có thể là giá trị hằng, biến, biểu thức hay kết hợp các dạng này
 - Trong lời gọi hàm, đối số thường được gọi là “đối số thực sự”
 - Bởi chúng chứa “dữ liệu thực sự” được truyền vào

Ví dụ về hàm:

Display 3.5 Khai báo, Định nghĩa, Lời gọi hàm (1/2)

Display 3.5

```
1  #include <iostream>
2  using namespace std;


3  double totalCost(int numberParameter, double priceParameter);
4  //Computes the total cost, including 5% sales tax,
5  //on numberParameter items at a cost of priceParameter each.

6  int main( )
7  {
8      double price, bill;
9      int number;

10     cout << "Enter the number of items purchased: ";
11     cin >> number;
12     cout << "Enter the price per item $";
13     cin >> price;

14     bill = totalCost(number, price);
```

*Function declaration;
also called the function
prototype*



Function call



Ví dụ về hàm:

Display 3.5 Khai báo, Định nghĩa, Lời gọi hàm (2/2)

```
15     cout.setf(ios::fixed);
16     cout.setf(ios::showpoint);
17     cout.precision(2);
18     cout << number << " items at "
19         << "$" << price << " each.\n"
20         << "Final bill, including tax, is $" << bill
21         << endl;

22     return 0;
23 }

24 double totalCost(int numberParameter, double priceParameter)
25 {
26     const double TAXRATE = 0.05; //5% sales tax
27     double subtotal;

28     subtotal = priceParameter * numberParameter;
29     return (subtotal + subtotal*TAXRATE);
30 }
```

*Function
head*

*Function
body*

*Function
definition*

SAMPLE DIALOGUE

Enter the number of items purchased: 2
Enter the price per item: \$10.10
2 items at \$10.10 each.
Final bill, including tax, is \$21.21

Một kiểu khai báo hàm khác

- Nhắc lại: Khai báo hàm là cung cấp “thông tin” cho trình biên dịch
- Trình biên dịch chỉ cần biết:
 - Kiểu trả về
 - Tên hàm
 - Danh sách tham số
- Ta có thể bỏ qua tên của tham số hình thức:
`double totalCost(int, double);`
 - Nếu cung cấp cả tên của tham số hình thức thì mã nguồn dễ đọc hơn

Phân biệt: Tham số và Đối số

- Tiếng Anh
 - tham số: parameter
 - đối số: argument
- Hai khái niệm này thường được dùng thay thế cho nhau
- Tham số/đối số hình thức
 - Dùng trong khai báo hàm
 - Dùng trong header của định nghĩa hàm
- Tham số/đối số thực sự
 - Dùng trong lời gọi hàm
- Máy móc mà nói thì tham số là “hình thức” còn đối số là “thực sự”
 - Không phải mọi tài liệu đều thống nhất dùng như vậy

Hàm gọi hàm

- Chúng ta đã và đang làm việc này rồi!
 - `main()` là một hàm!
- Yêu cầu duy nhất:
 - Khai báo của hàm phải xuất hiện trước lời gọi hàm
- Định nghĩa hàm có thể nằm ở đâu đó
 - Phía dưới định nghĩa hàm `main()`
 - Hoặc trong một tệp riêng biệt
- Thường thì một hàm gọi tới rất nhiều hàm khác
- Thậm chí hàm có thể gọi tới chính nó → “đệ quy”

Hàm trả về giá trị logic

- Kiểu trả về của hàm có thể là bất cứ kiểu hợp lệ nào
 - Giả sử đã biết khai báo/nguyên mẫu hàm:
`bool appropriate(int rate);`
 - Và định nghĩa hàm:

```
bool appropriate (int rate)
{
    return (((rate>=10)&&(rate<20))||(rate==0));
}
```
 - Hàm này sẽ trả về "true" hoặc "false"
 - Ta có thể gọi hàm này ở bên trong hàm khác như sau:

```
if (appropriate(entered_rate))
    cout << "Rate is valid\n";
```

Khai báo hàm void

- Tương tự như các hàm có trả về giá trị
- Chỉ định kiểu trả về của hàm là "void"
- Ví dụ:
 - Khai báo/nguyên mẫu hàm:

```
void showResults(           double fDegrees,  
                    double cDegrees);
```

 - Kiểu trả về là "void"
 - Không có gì được trả về

Định nghĩa hàm void

- Định nghĩa hàm:

```
void showResults(double fDegrees, double cDegrees)
{
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);
    cout << fDegrees
         << " degrees fahrenheit equals \n"
         << cDegrees << " degrees celsius.\n";
}
```

- Chú ý: không có lệnh `return`
 - Đối với hàm `void`, lệnh `return` là tùy chọn

Gọi tới hàm void

- Cũng giống như gọi tới các hàm có sẵn trong thư viện
- Gọi từ hàm khác, ví dụ `main()`:
 - `showResults(degreesF, degreesC);`
 - `showResults(32.5, 0.3);`
- Chú ý là không gọi trong phép gán bởi không có giá trị trả về
- Đối số thực sự (`degreesF, degreesC`)
 - Được truyền vào hàm
 - Hàm được gọi để “làm công việc của nó” với những dữ liệu truyền vào

Bàn thêm về lệnh return

- Trả điều khiển cho hàm gọi tới hàm này
 - Nếu kiểu trả về khác void, hàm PHẢI có lệnh **return**
 - Thường thì return là lệnh cuối cùng trong định nghĩa hàm
- Đối với hàm **void**, lệnh **return** là tùy chọn
 - Dấu đóng ngoặc nhọn “}” là cách **return** không tường minh cho hàm **void**

Điều kiện trước và điều kiện sau

- Tiếng Anh
 - điều kiện trước: precondition
 - điều kiện sau: postcondition
- Tương tự như thảo luận về "I-P-O"
- Cách chú thích trong khai báo hàm:
`void showInterest(double balance, double rate);`
//Điều kiện trước: balance is nonnegative account balance
// rate is interest rate as percentage
// Điều kiện sau: amount of interest on given balance,
// at given rate ...
- Thường gọi là Input & Output

main(): “Đặc biệt”

- Nhắc lại: `main()` là một hàm
- “Đặc biệt” ở chỗ:
 - Có một và chỉ một hàm `main()` tồn tại trong một chương trình
- Cái gì gọi hàm `main()`?
 - Hệ điều hành
 - Theo truyền thống nó nên có lệnh `return`
 - Giá trị này được truyền tới “nơi gọi hàm `main`” → Ở đây là hệ điều hành
 - Nên trả về `"int"` hoặc `"void"`

Quy tắc về phạm vi hoạt động

- Biến cục bộ
 - Khai báo trong thân một hàm
 - Chỉ hoạt động trong hàm đó
- Ta có thể có những biến cùng tên khai báo trong hàm khác
 - Phạm vi ở đây là cục bộ: “phạm vi của một biến là bên trong hàm nó được khai báo”
- Biến cục bộ
 - Duy trì điều khiển trên dữ liệu đơn lẻ
 - Hàm nên khai báo tất cả dữ liệu cục bộ nó cần

Trừu tượng hóa thủ tục

- Người sử dụng chỉ cần biết một hàm “làm gì” chứ không cần biết các bước nó thực hiện việc đó “như thế nào”.
- Hãy nghĩ tới “hộp đen”
 - Là thiết bị bạn biết cách sử dụng chứ không biết chi tiết hoạt động bên trong của nó
- Cài đặt hàm giống như hộp đen
 - Người dùng hàm chỉ cần biết khai báo
 - Không cần biết định nghĩa hàm
 - Gọi là Che giấu Thông tin
 - Giấu đi chi tiết về các bước hàm thực hiện công việc của nó

Hằng toàn cục và Biến toàn cục

- Biến/hằng được khai báo “bên ngoài” tất cả các hàm
 - Là toàn cục với tất cả các hàm trong tệp đó
- Biến/hằng được khai báo “bên trong” thân hàm
 - Là cục bộ với hàm đó
- Khai báo toàn cục thường sử dụng với hằng:
 - `const double TAXRATE = 0.05;`
 - Ta khai báo toàn cục để tất cả các hàm đều có thể truy cập tới hằng
- Biến toàn cục?
 - Có thể nhưng hiếm khi sử dụng
 - Nguy hiểm: Không kiểm soát được các hàm sẽ dùng nó như thế nào!

Khối

- Khai báo dữ liệu bên trong lệnh khối
 - Dữ liệu sẽ hoạt động trong phạm vi khối
- Chú ý: mỗi định nghĩa hàm là một khối!
 - Đây chính là “phạm vi hoạt động hàm” cục bộ

- Khối lặp:

```
for (int ctr=0;ctr<10;ctr++)  
{  
    sum+=ctr;  
}
```

- Biến ctr chỉ hoạt động trong khối vòng lặp này

Các phạm vi lồng nhau

- Các biến cùng tên được khai báo trong nhiều khối
- Hợp lệ; phạm vi ở đây là “phạm vi khối”
 - Không nhập nhằng
 - Mỗi tên là duy nhất trong phạm vi của nó

Tóm tắt 1

- Hai loại hàm:
 - Có trả về giá trị
 - Hàm void
- Hàm nên được xem như “hộp đen”
 - Che đi chi tiết từng bước nó giải quyết vấn đề
 - Khai báo dữ liệu cục bộ nó cần
- Khai báo hàm nên chú thích đủ hướng dẫn sử dụng
 - Điều kiện trước và sau
 - Những tiêu chuẩn cho nơi gọi hàm

Tóm tắt 2

- Dữ liệu cục bộ
 - Khai báo trong định nghĩa hàm
- Dữ liệu toàn cục
 - Khai báo bên trên tất cả các định nghĩa hàm
 - Dùng được với hằng, không nên dùng cho biến
- Tham số/ Đối số
 - Hình thức: Trong khai báo và định nghĩa hàm
 - Là “chỗ đặt trước” cho dữ liệu truyền vào
 - Thực sự: Trong lời gọi hàm
 - Là dữ liệu thực sự truyền cho hàm

Chuẩn bị bài tới

- Đọc chương 4 giáo trình: Tham số của hàm và Nạp chồng hàm