Ngôn ngữ lập trình

Bài 3:
Hàm và Nạp chồng Hàm

Giảng viên: Lê Nguyễn Tuấn Thành
Email: thanhlnt@tlu.edu.vn

Bộ Môn Công Nghệ Phần Mềm – Khoa CNTT
Trường Đại Học Thủy Lợi

Nội dung

1. Hàm (Function)
2. Nạp chồng hàm (Overloading)

Bài giảng có sử dụng hình vẽ trong cuốn sách "*Absolute C++*. W. Savitch, Addison Wesley, 2002"

1. HÀM

Function

Cơ bản về hàm

- Hàm được định nghĩa sẵn
  - Hàm trả về một giá trị
  - Hàm không trả về giá trị nào (hàm void)
- Hàm do người dùng định nghĩa
  - Khai báo, định nghĩa, gọi hàm
  - Hàm đệ quy (recursive functions)
- Quy tắc phạm vi (scope rules)
  - Biến địa phương (local)
  - Hằng số (constant) và biến toàn cục (global)
  - Khối, phạm vi lồng nhau (nested scopes)

Giới thiệu về hàm

- Hàm (Function): *một khối của chương trình (blocks of programs) có mục đích rõ ràng*
- Một số thuật ngữ (cách gọi) khác của hàm trong những ngôn ngữ khác:
  - Phương thức (*procedures*), chương trình con (*subprograms*), phương thức (*methods*)
- Khái niệm **I – P – O**
  - Input – Process – Output
  - Thành phần cơ bản của bất kỳ chương trình nào
  - Thao tác với hàm dựa trên các thành phần của khái niệm này

Hàm định nghĩa trước (Predefined functions)

- C++ cung cấp nhiều thư viện với đầy đủ các hàm được định nghĩa sẵn !
- Hai kiểu:
  - Hàm trả về một giá trị
  - Hàm không trả về giá trị nào
- Phải "*#include*" thư viện thích hợp
  - Ví dụ: <cmath>, <cstdlib>, <iostream> …

Sử dụng hàm định nghĩa sẵn

- C++ có vô số các hàm toán học định nghĩa sẵn!
  - Trong thư viện <*cmath.h*>
  - Hầu hết trả về một giá trị
- Ví dụ: ***theRoot = sqrt(9.0);***
  - Các thành phần của biểu thức trên:
    - **sqrt**: tên hàm
    - **theRoot**: biến được gán giá trị trả về của hàm
    - **9.0**: đối số (*argument*) hoặc đầu vào (*starting input*) cho hàm
  - Viết theo khái niệm I – P – O:
    - I = 9.0
    - P = "tính căn bậc hai" (the square root)
    - O = 3.0, giá trị trả về của hàm, được gán cho biến *theRoot*

Lời gọi hàm (Function call)

- Trở lại ví dụ trước *theRoot = sqrt(9.0);*
  - Biểu thức *sqrt(9.0)* là một lời gọi hàm (*function call* hay *function invocation*)
  - Đối số của một lời gọi hàm có thể là một literal (vd: 9.0), một biến hay một biểu thức
  - Lời gọi hàm có thể được sử dụng trong một biểu thức khác (ví dụ: *bonus = sqrt(sales)/10;*)

Chương trình với hàm định nghĩa sẵn (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);
```

Chương trình với hàm định nghĩa sẵn (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.
```

Một số hàm định nghĩa sẵn khác (1/3)

- **#include <cstdlib>**
  - *abs()*: trả về giá trị tuyệt đối của một số nguyên (int)
  - *labs()*: trả về giá trị tuyệt đối của một số nguyên lớn (long int)
  - *fabs()*: trả về giá trị tuyệt đối của một số thực (float)
- Hàm toán học
  - *pow (x, y)*: x^y
- Lưu ý: một hàm có thể nhiều đối số, mỗi đối số có thể có kiểu dữ liệu khác nhau

Một số hàm định nghĩa sẵn khác (2/3)

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 int	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Absolute value for long	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Absolute value for double	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath

Một số hàm định nghĩa sẵn khác (3/3)

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

Hàm **void** định nghĩa sẵn

- Không có giá trị trả lại
- Hàm **void** vẫn có thể có đối số (arguments)
- Ví dụ: *exit (1)*

Bộ tạo số ngẫu nhiên

- Trả về một số "*được chọn ngẫu nhiên*"
- Được sử dụng cho mô phỏng (simulation), trò chơi (game)
  - *rand()*: trả về một giá trị ở giữa 0 và RAND_MAX
  - Scaling: ép buộc các số ngẫu nhiên vào một khoảng nhỏ hơn. Ví dụ: *rand() % 6 // trả lại giá trị ngẫu nhiên trong khoảng từ 0 đến 5*
  - Shifting: *rand() % 6 + 1*
- Hạt giống (seed) dùng để tạo số ngẫu nhiên
  - Hàm *rand()* tạo ra một chuỗi trình tự (sequence) các số ngẫu nhiên
  - Chúng ta có thể sử dụng "hạt giống (seed)" để thay thế trình tự tạo số ngẫu nhiên với hàm ***srand(seed_value)***:
    - void function
    - Seed có thể là bất kỳ giá trị nào. Ví dụ: ***srand(time(0));***

Bài tập

- Viết chương trình C++ sinh ra N số ngẫu nhiên (N < 100) trong khoảng 0 đến 1000, sau đó sắp xếp theo thứ tự tăng dần hoặc giảm dần

  Gợi ý: sử dụng hàm sẵn có ***rand()***

Hàm do người dùng định nghĩa

- Lập trình viên tự viết hàm cho mục đích của mình!
- Xây dựng những khối của chương trình
  - Chia để trị (Divide & Conquer)
  - Dễ đọc (Readability)
  - Sử dụng lại (Re-use)
- Hàm mà bạn định nghĩa có thể:
  - Trong cùng một file với hàm *main*()
  - Ở file khác, riêng biệt để người khác có thể dùng nó

Những bước cần có khi xây dựng một hàm

- 3 bước khi xây dựng hàm
  1. **Khai báo** / nguyên mẫu hàm (function declaration/prototype)
     - Thông tin cho trình biên dịch (compiler)
     - Thông dịch (interpret) thích hợp lời gọi hàm
  2. **Định nghĩa hàm** (function definition)
     - Cài đặt thực tế của hàm
  3. **Gọi hàm** (function call)
     - Chuyển điều khiển cho hàm

Khai báo hàm (Function declaration)

- Còn được gọi là nguyên mẫu hàm (function prototype)
- Khai báo thông tin cho trình biên dịch. Nói cho trình biên dịch biết cách để thông dịch lời gọi hàm
  - Cú pháp:

<giá_trị_trả_lại> TênHàm(danh_sách_đối_số);

  - Ví dụ:

double totalCost(int numberParameter, double priceParameter);

- Được đặt trước bất kỳ lời gọi hàm
  - Trong không gian khai báo của hàm main()
  - Hoặc ở trước hàm main() trong không gian toàn cục

Định nghĩa hàm (Function definition)

- Cài đặt thực tế của hàm. Giống như cài đặt của hàm *main()*
- Ví dụ:

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

- Được đặt sau hàm *main*(). Không đặt bên trong hàm *main*()
- Lệnh *return*: trả quyền điều khiển ngược lại đối tượng gọi hàm

Lời gọi hàm (Function call)

- Giống như gọi những hàm đã được định nghĩa sẵn
- Ví dụ: *bill = totalCost(number, price);* // totalCost trả lại một giá trị kiểu double và giá trị này được gán cho biến *bill*
  - Hai tham số: *number, price*

Ví dụ hàm người dùng định nghĩa (1/2)

Display 3.5 A Function Using a Random Number Generator

```
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

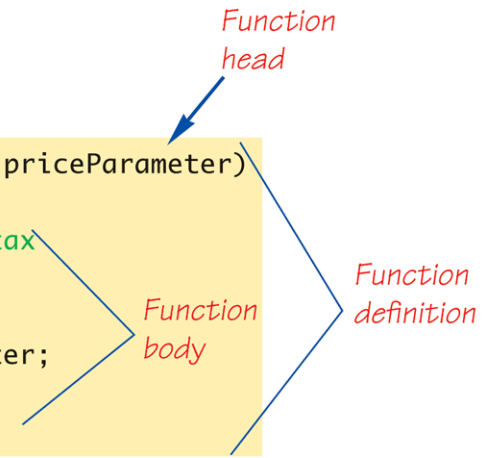
Ví dụ hàm người dùng định nghĩa (2/2)

```
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  }
```



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

Hàm trả lại giá trị kiểu **boolean**

1. Khai báo hàm:

```
bool appropriate(int rate);
```

2. Định nghĩa hàm:

```
bool appropriate (int rate)
{
        return (((rate>=10)&&(rate<20))||(rate==0);
}
```

3. Gọi hàm:

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

Khai báo hàm **void**

- Giống như khai báo hàm trả về giá trị
- Nhưng kiểu trả về là "**void**".
- Ví dụ khai báo hàm sau:

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

- Định nghĩa hàm. Không có lệnh *return*

```
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";
}
```

- Gọi giống như những hàm *void* định nghĩa sẵn. Không có phép gán

```
showResults(32.5, 0.3);
```

Điều kiện trước và điều kiện sau

- Tương tự như khái niệm I – P – O
- Chú thích trong khai báo hàm

```
void showInterest(double balance, double rate);
// Precondition: balance is nonnegative account balance
//               rate is interest rate as percentage
// Postcondition: amount of interest on given balance,
//                at given rate …
```

- Cũng thường được gọi là *Input* và *Output*

Hàm đặc biệt **main**()

- ***main()*** cũng là một hàm NHƯNG chỉ có duy nhất một hàm *main()* trong chương trình
- Ai gọi hàm *main()* ?
  - Hệ điều hành
  - Thông thường có một lệnh return để trả giá trị về cho đối tượng gọi (caller), ở đây là hệ điều hành
  - Kiểu trả lại thường là *int* hoặc *void*

Quy tắc phạm vi (scope rules)

- Biến địa phương (local variables)
  - Được khai báo trong thân hàm
  - Chỉ được sử dụng bên trong hàm đó
  - Có thể khai báo biến cùng tên trong những hàm khác nhau
- Biến toàn cục (global variables)
  - Khai báo bên ngoài thân hàm
  - Thường cho các hằng số

Tóm tắt cho phần hàm

- Hai kiểu hàm
  - Hàm trả về giá trị và hàm void
- Hàm giống như hộp đen (black boxes)
  - Ẩn đi chi tiết của việc "làm sao để làm được điều đó" (how)
  - Khai báo dữ liệu địa phương của nó
- Khai báo hàm nên được chú thích đầy đủ
  - Chú thích điều kiện trước và sau (pre- & post-conditions)
  - Chú thích tất cả đối tượng gọi hàm này
- Biến địa phương
  - Khai báo bên trong định nghĩa hàm
- Biến toàn cục
  - Khai báo bên trên định nghĩa hàm
  - Thường cho hằng số, không nên cho các biến
- Tham số / Đối số (Parameters/Arguments)

Bài tập về hàm

- Viết một hàm để tính giai thừa của một số nguyên dương.

2. Tham số và nạp chồng hàm

Tham số và nạp chồng hàm

- Tham số
  - Tham trị (call-by-value)
  - Tham chiếu (call-by-reference)
  - Danh sách tham số trộn lẫn
- Nạp chồng và đối số mặc định
  - Ví dụ, Quy tắc
- Test và gỡ rối (debug) hàm
  - assert macro
  - stubs, drivers

Tham số

- 2 cách thức truyền tham số cho hàm

1. **Truyền tham trị (call-by-value)**
   - Bản sao có giá trị giống tham số được truyền vào hàm
   - Được xem như biến địa phương của hàm
   - Nếu thay đổi, chỉ có bản sao địa phương (local copy) thay đổi. Hàm không có quyền truy cập vào tham số thực sự của đối tượng gọi hàm
2. **Truyền tham chiếu (call-by-reference)**
   - Địa chỉ của tham số được truyền vào hàm
   - Cung cấp quyền truy cập tới tham số thực sự của đối tượng gọi hàm
   - Dữ liệu có thể được thay đổi bởi hàm
   - Thêm dấu **&** (ampersand) vào trước tham số

Chương trình với Tham trị (call-by-value) (1/3)

Display 4.1 Formal Parameter Used as a Local Variable

```
1   //Law office billing program.
2   #include <iostream>
3   using namespace std;

4   const double RATE = 150.00; //Dollars per quarter hour.

5   double fee(int hoursWorked, int minutesWorked);
6   //Returns the charges for hoursWorked hours and
7   //minutesWorked minutes of legal services.

8   int main( )
9   {
10      int hours, minutes;
11      double bill;
```

Chương trình với Tham trị (call-by-value) (2/3)

```
12      cout << "Welcome to the law office of\n"
13           << "Dewey, Cheatham, and Howe.\n"
14           << "The law office with a heart.\n"
15           << "Enter the hours and minutes"
16           << " of your consultation:\n";
17      cin >> hours >> minutes;

18      bill = fee(hours, minutes);

19      cout.setf(ios::fixed);
20      cout.setf(ios::showpoint);
21      cout.precision(2);
22      cout << "For " << hours << " hours and " << minutes
23           << " minutes, your bill is $" << bill << endl;

24      return 0;
25  }
```

The value of `minutes` *is not changed by the call to* `fee`.

(continued)

Chương trình với Tham trị (call-by-value) (3/3)

Display 4.1 Formal Parameter Used as a Local Variable

```
26  double fee(int hoursWorked, int minutesWorked)
27  {
28      int quarterHours;

29      minutesWorked = hoursWorked*60 + minutesWorked;
30      quarterHours = minutesWorked/15;
31      return (quarterHours*RATE);
32  }
```

`minutesWorked` *is a local variable initialized to the value of* `minutes`.

SAMPLE DIALOGUE

Welcome to the law office of
Dewey, Cheatham, and Howe.
The law office with a heart.
Enter the hours and minutes of your consultation:
5 46
For 5 hours and 46 minutes, your bill is $3450.00

Chương trình với Tham chiếu (call-by-reference) (1/3)

Display 4.2 **Call-by-Reference Parameters**

```
1   //Program to demonstrate call-by-reference parameters.
2   #include <iostream>
3   using namespace std;

4   void getNumbers(int& input1, int& input2);
5   //Reads two integers from the keyboard.

6   void swapValues(int& variable1, int& variable2);
7   //Interchanges the values of variable1 and variable2.

8   void showResults(int output1, int output2);
9   //Shows the values of variable1 and variable2, in that order.

10  int main( )
11  {
12      int firstNum, secondNum;

13      getNumbers(firstNum, secondNum);
14      swapValues(firstNum, secondNum);
15      showResults(firstNum, secondNum);
16      return 0;
17  }
```

Chương trình với Tham chiếu (call-by-reference) (2/3)

```
18  void getNumbers(int& input1, int& input2)
19  {
20      cout << "Enter two integers: ";
21      cin >> input1
22          >> input2;
23  }

24  void swapValues(int& variable1, int& variable2)
25  {
26      int temp;

27      temp = variable1;
28      variable1 = variable2;
29      variable2 = temp;
30  }
31
32  void showResults(int output1, int output2)
33  {
34      cout << "In reverse order the numbers are: "
35           << output1 << " " << output2 << endl;
36  }
```

Chương trình với Tham chiếu (call-by-reference) (3/3)

Display 4.2 **Call-by-Reference Parameters**

SAMPLE DIALOGUE

```
Enter two integers: 5 6
In reverse order the numbers are: 6 5
```

Phân biệt Tham số và đối số

- Tham số (Parameters) vs Đối số (Arguments)
  - Gọi là **tham số** khi khai báo và định nghĩa hàm (bước 1, 2)
  - Gọi là **đối số** khi gọi hàm (bước 3)
- Danh sách tham số trộn lẫn. Ví dụ:

  void mixedCall(int& par1, int par2, double& par3);

  Khi gọi hàm *mixedCall(arg1, arg2, arg3);*
  - *arg1:* là 1 kiểu int, được truyền vào bởi tham chiếu
  - *arg2:* là 1 kiểu int, được truyền vào bởi tham trị
  - *arg3:* là 1 kiểu double, được truyền vào bởi tham chiếu

Nạp chồng hàm (1/2)

- Khai báo nhiều tên hàm giống nhau, chỉ khác nhau danh sách tham số.
- Phân biệt các hàm này bằng cặp
  <*tên_hàm, danh_sách_tham_số*>, được gọi là **chữ ký** (signature)
- ~~Định nghĩa của các hàm này khác nhau~~
- Ví dụ: hai hàm tính giá trị trung bình
  - Của 2 tham số

```
double average(double n1, double n2)
{
    return ((n1 + n2) / 2.0);
}
```

  - Của 3 tham số

```
double average(double n1, double n2, double n3)
{
    return ((n1 + n2 + n3) / 3.0);
}
```

Nạp chồng hàm (2/2)

- Hàm nào sẽ được gọi?
- Phụ thuộc vào danh sách đối số
- Phân giải nạp chồng hàm (overloading resolution):
  - Trùng khớp chính xác: tìm kiếm một hàm với chữ ký trùng khớp chính xác (exact signature)
  - Trùng khớp tương thích (compatible match): tìm kiếm một hàm với chữ ký trùng khớp tương thích ("compatible" signature) khi sự ép kiểu tự động có thể thực thi
    - Ép kiểu lên (promotion, vd: int -> double): không mất dữ liệu
    - Ép kiểu xuống (demotion, vd: double -> int): có thể mất dữ liệu
- Ví dụ phân giải nạp chồng hàm:
  - *1. void f (int n, double m);*
    2. void f (double n, int m);
    3. void f (int n, int m);
  - *f(98, 99);* → gọi hàm số 3
    f(5.3, 4); → gọi hàm số 2
    f(4.3, 5.2); → gọi hàm số mấy ???

Ép kiểu tự động và nạp chồng

- Cho phép các kiểu số khác tự động chuyển thành kiểu "*double*" khi cần (*int -> double, float -> double, char -> double*)
- Ví dụ:

```
double mpg(double miles, double gallons)
{
        return (miles/gallons);
}
```

- Lời gọi hàm
  - *mpgComputed = mpg(5, 20);* // chuyển 5 và 20 thành kiểu double, sau đó truyền vào hàm
  - *mpgComputed = mpg(5.8, 20.2);* // không cần thiết phải chuyển kiểu
  - *mpgComputed = mpg(5, 2.4);* // chuyển 5 thành 5.0, sau đó truyền vào hàm

Đối số mặc định (Defaut argument)

- Cho phép gọi hàm thiếu một số đối số
- Dùng giá trị mặc định của tham số khi khai báo / định nghĩa hàm
- Ví dụ:

```
void showVolume (int length, int width = 1, int height = 1);
```

- Những lời gọi hàm sau hợp lệ:
  - *showVolume(2,4,6);*
  - *showVolume(3,5);* // thiếu đối số height, height lấy giá trị mặc định là 1
  - *showVolume(7);* // thiếu 2 đối số width & height, lấy giá trị mặc định là 1

Chương trình đối số mặc định (1/2)

Display 4.8 **Default Arguments**

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

4   void showVolume(int length, int width = 1, int height = 1);
5   //Returns the volume of a box.
6   //If no height is given, the height is assumed to be 1.
7   //If neither height nor width is given, both are assumed to be 1.

8   int main( )
9   {
10      showVolume(4, 6, 2);
11      showVolume(4, 6);
12      showVolume(4);

13      return 0;
14  }

15  void showVolume(int length, int width, int height)
```

Default arguments

A default argument should not be given a second time.

Chương trình đối số mặc định (2/2)

```
16  {
17      cout << "Volume of a box with \n"
18           << "Length = " << length << ", Width = " << width << endl
19           << "and Height = " << height
20           << " is " << length*width*height << endl;
21  }
```

SAMPLE DIALOGUE

```
Volume of a box with
Length = 4, Width = 6
and Height = 2 is 48
Volume of a box with
Length = 4, Width = 6
and Height = 1 is 24
Volume of a box with
Length = 4, Width = 1
and Height = 1 is 4
```

Bài tập nạp chồng hàm

- Sử dụng nạp chồng hàm để xếp thứ tự 10 số kiểu int, hoặc 10 giá trị long hoặc 10 giá trị double trong cùng một chương trình C++.

Test và debug hàm

- Có nhiều cách thức khác nhau để kiểm tra tính chính xác của một hàm tự định nghĩa
  1. Dùng lệnh *cout* để in kết quả ra màn hình trong lúc định nghĩa và gọi hàm
  2. Sử dụng một bộ gỡ rối của trình biên dịch *(compiler debugger)*
  3. Sử dụng *assert macro*
  4. Sử dụng *stubs* và *drivers*

Assert macro

- *Assertion*: một câu lệnh trả về true hoặc false
- Sử dụng để kiểm tra độ chính xác của điều kiện
  - Cú pháp: *assert(<assert_condition>);*
  - Không có giá trị trả lại
  - Đánh giá *assert_condition*, kết thúc nếu *false*, tiếp tục nếu *true*
- Được định nghĩa trong thư viện ***<cassert>***

Ví dụ về assert macro

- Khai báo hàm

  void computeCoin (int coinValue, int& number, int& amountLeft);
  //Precondition: 0 < coinValue < 100, 0 <= amountLeft <100
  //Postcondition: number set to max. number of coins

- Kiểm tra điều kiện trước
  - assert ((0 < coinValue) && (coinValue < 100)
    && (0 <= amountLeft) && (amountLeft < 100));
  - Nếu điều kiện trước không thỏa mãn -> điều kiện là false -> chương trình kết thúc ngay lập tức
- Hữu ích trong việc gỡ rối (debugging)

Bật/tắt assert

- *#define NDEBUG*
  #include <cassert>
- Thêm dòng *#define NDEBUG* trước dòng *#include* để tắt tất cả các assertions trong chương trình
- Xóa dòng này (hoặc tạo thành dòng chú thích) sẽ bật chức năng *assert* trong chương trình

Stubs và drivers

- Phân chia các đơn vị biên dịch
  - Mỗi hàm được thiết kế (design), code, test riêng biệt
  - Đảm bảo tính hợp lệ của từng đơn vị
  - Chia để trị: chuyển các nhiệm vụ lớn thành các tác vụ nhỏ hơn, dễ quản lý hơn
- NHƯNG làm sao để test các hàm độc lập nhau
  - Dùng những chương trình driver

Ví dụ chương trình **driver** (1/3)

Display 4.9 **Driver Program**

```
1
2   //Driver program for the function unitPrice.
3   #include <iostream>
4   using namespace std;

5   double unitPrice(int diameter, double price);
6   //Returns the price per square inch of a pizza.
7   //Precondition: The diameter parameter is the diameter of the pizza
8   //in inches. The price parameter is the price of the pizza.

9   int main( )
10  {
11      double diameter, price;
12      char ans;

13      do
14      {
15          cout << "Enter diameter and price:\n";
16          cin >> diameter >> price;
```

Ví dụ chương trình **driver** (2/3)

```
17          cout << "unit Price is $"
18               << unitPrice(diameter, price) << endl;

19          cout << "Test again? (y/n)";
20          cin >> ans;
21          cout << endl;
22      } while (ans == 'y' || ans == 'Y');

23      return 0;
24  }
25
26  double unitPrice(int diameter, double price)
27  {
28      const double PI = 3.14159;
29      double radius, area;

30      radius = diameter/static_cast<double>(2);
31      area = PI * radius * radius;
32      return (price/area);
33  }
```

(continued)

Ví dụ chương trình **driver** (3/3)

Display 4.9 **Driver Program**

SAMPLE DIALOGUE

Enter diameter and price:
13 14.75
`Unit price is: $0.111126`
Test again? (y/n): **y**

Enter diameter and price:
2 3.15
`Unit price is: $1.00268`
Test again? (y/n): **n**

Stubs

- Phát triển chương trình từng bước
- Viết các hàm lớn trước (big-picture)
- Viết các hàm cấp thấp (low-level) sau
- Stub-out các hàm cho đến khi cài đặt
  - Ví dụ:

```
double unitPrice(int diameter, double price)
{
    return (9.99); // not valid, but noticeably a "temporary" value
}
```

  - Lời gọi tới hàm này vẫn hoạt động

Quy tắc test căn bản

- Viết chương trình đúng
- Tối giảm hóa lỗi (errors), bugs
- Đảm bảo tính hợp lệ của dữ liệu
  - Test và debug từng hàm trong chương trình một cách lần lượt
  - Tránh lỗi phân tầng (error-cascading) và kết quả xung đột (conflicting results)

Tóm tắt cho phần nạp chồng hàm

- Tham trị (call-by-value) là những bản sao cục bộ (local copies) trong thân hàm. Đối số thực sự của hàm không thể bị thay đổi
- Tham chiếu (call-by-reference) truyền địa chỉ bộ nhớ của đối số thực sự vào hàm. Đối số thực sự có thể bị thay đổi
- C++ cho phép nhiều định nghĩa của hàm cùng một tên hàm: gọi là nạp chồng hàm
- Tham số mặc định cho phép lời gọi hàm thiếu một vài đối số trong danh sách truyền vào. Khi đó các giá trị mặc định sẽ được sử dụng
- assert macro sẽ kết thúc chương trình nếu điều kiện kiểm tra là false
- Các hàm nên được test một cách độc lập, như các đơn vị biên dịch riêng biệt và với driver

Tham khảo

- **Giáo trình chính: W. Savitch, *Absolute C++*, Addison Wesley, 2002**
- Tham khảo:
  - A. Ford and T. Teorey, *Practical Debugging in C++*, Prentice Hall, 2002
  - Nguyễn Thanh Thủy, *Kĩ thuật lập trình C++*, NXB Khoa học và Kĩ Thuật, 2006