Nhập Môn Lập Trình
Hàm – Cơ Bản

TS. Lê Nguyên Khôi
Trường Đại học Công nghệ, ĐHQGHN

Nội Dung

- Khái niệm chung
- Định nghĩa hàm
- Sử dụng hàm
- Truyền tham số
- Phạm vi biến

Một Số Ví Dụ

- Hàm đã định nghĩa (có sẵn trong thư viện)
  - `sqrt`
  - `pow`

- Hàm tự định nghĩa
  - `main`
  - `f`
  - `tong`
  - `hieu`

Hàm Thư Viện **<cmath>**

- Hàm tính căn bậc hai **sqrt**
- Hàm tính số mũ **pow**

```
#include <iostream>
#include <cmath>

int main()
{
    double a = 4.0;      // a = 3.0;
    double b = sqrt(a);
    double c = pow(b, 2);
    cout << (a == c) << endl;
    return 0;
}
```

Hàm Tự Định Nghĩa – $f(a,b)$

$$f(a,b) = a^3 + 2a^2b + 3ab^2 + 4b^3$$

- Tên hàm: **f**, danh sách biến số: **a**, **b**
- Kiểu dữ liệu hàm: **int**, biến số: **int**, **int**
- Câu lệnh **return** trả về giá trị của hàm

```
int f(int a, int b)      // f(a=2,b=1) = 26
{
    return a*a*a + 2*a*a*b + 3*a*b*b + 4*b*b*b;
}

int main() {
    cout << f(2, 1) << endl;  // in ra 26
    return 0;
}
```

Hàm Tự Định Nghĩa – $\text{tong}(a, b)$

$$\text{tong}(a, b) = a + b$$

- Hàm tính tổng hai số nguyên
- Tên hàm: **tong**, danh sách biến số: **a**, **b**
- Kiểu dữ liệu hàm: **int**, biến số: **int**, **int**

```
int tong(int a, int b) { return (a + b); }

int main() {
    int soThu1; cin >> soThu1;
    int soThu2; cin >> soThu2;
    int soTong = tong(soThu1, soThu2);
    cout << tong(soThu2, soThu1) << endl;
    return 0;
}
```

Hàm Tự Định Nghĩa – hieu(a, b)

$$\text{hieu}(a, b) = a - b$$

- Hàm tính hiệu hai số nguyên
- Tên hàm: **`hieu`**, danh sách biến số: **`a`**, **`b`**
- Kiểu dữ liệu hàm: **`int`**, biến số: **`int`**, **`int`**

```
int hieu(int a, int b) { return (a - b); }

int main() {
    int soThu1; cin >> soThu1;
    int soThu2; cin >> soThu2;
    int soHieu = hieu(soThu1, soThu2);
    cout << soHieu << endl;
    return 0;
}
```

Hàm Tự Định Nghĩa

- Hàm có kiểu (**int**, **double**, **bool**), giống biến
- Hàm có tên, cách đặt tên giống biến
- Hàm cần dữ liệu để xử lý
- Kiểu của hàm giống kiểu của dữ liệu trả về

```
int f(int a, int b)
{
    return a*a*a + 2*a*a*b + 3*a*b*b + 4*b*b*b;
}

int tong(int a, int b) { return (a + b); }

int hieu(int a, int b) { return (a - b); }
```

Hàm Tự Định Nghĩa – chiaHet(a, b)

- Kiểm tra 2 số có chia hết cho nhau không ? chiaHet(a, b)

```
#include <iostream>

bool chiaHet(int a, int b) // a = 4, b = 2
{
    bool ket_qua = false;
    return ket_qua;
}
int main()
{
    cout << chiaHet(4, 2) << endl;
    return 0;
}
```

Hàm Tự Định Nghĩa – chiaHet(a, b)

- Kiểm tra 2 số có chia hết cho nhau không ? chiaHet(a, b)

```
bool chiaHet(int a, int b) // a = 4, b = 2
{
    bool ket_qua = false;
    if (a % b == 0) ket_qua = true;
    else ket_qua = false;
    return ket_qua;
}
int main()
{
    cout << chiaHet(4, 2) << endl;
    return 0;
}
```

Hàm Tự Định Nghĩa – chiaHet(a, b)

- Kiểm tra 2 số có chia hết cho nhau không ? chiaHet(a, b)

```
bool chiaHet(int a, int b) // a = 2, b = 4
{
    bool ket_qua = false;
    if (a%b == 0 || b%a == 0) ket_qua = true;
    else ket_qua = false;
    return ket_qua;
}
int main()
{
    cout << chiaHet(2, 4) << endl;
    return 0;
}
```

Hàm Tự Định Nghĩa – chiaHet(a, b)

- Kiểm tra 2 số có chia hết cho nhau không ? chiaHet(a, b)

```
bool chiaHet(int a, int b) // a = 2, b = 4
{
    bool ket_qua = false;
    ket_qua = (a % b == 0 || b % a == 0);
    return ket_qua;
}

int main()
{
    cout << chiaHet(2, 4) << endl;
    return 0;
}
```

Hàm Tự Định Nghĩa – chiaHet(a, b)

- Kiểm tra 2 số có chia hết cho nhau không ? chiaHet(a, b)

```
bool chiaHet(int a, int b) // a = 2, b = 4
{
    return (a % b == 0 || b % a == 0);
}
int main()
{
    cout << chiaHet(2, 4) << endl;
    return 0;
}
```

```
int tong(int a, int b) { return (a + b); }
int hieu(int a, int b) { return (a - b); }
```

Hàm Tự Định Nghĩa – Lũy Thừa X^y

- Trả về kết quả sau khi tính X mũ y
- Hàm thực hiện thao tác
  - Thực hiện phép nhân lặp đi lặp lại
  - Ví dụ: $2^4 = 16 = 2 * 2 * 2 * 2$
- Hàm cần dữ liệu để thực hiện
  - 2 số (**`int`**) đặt tên **`co_so`**, **`so_mu`**
- Hàm cần trả về kết quả sau khi thực hiện
  - 1 số (**`int`**) đặt tên **`ket_qua`**
- Tên **`luyThua`**
- Kiểu **`int`**

Hàm Tự Định Nghĩa – Lũy Thừa X^y

- Gọi **`luyThua(2,4)`** trả về kết quả **`16`**
- Gọi **`luyThua(2.0,4.0)`** trả về kết quả **`16`**
- Gọi **`luyThua(2.4,4.4)`** trả về kết quả **`16`**
- Gọi **`luyThua(2,-1)`** trả về kết quả

```
int luyThua(int co_so, int so_mu)
{
    int ket_qua = 1;
    for (int i = 1; i <= so_mu; i = i + 1)
    {
        ket_qua = ket_qua * co_so;
    }
    return ket_qua;
}
```

Bài Toán Phân Loại Tam Giác

- Cho độ dài 3 đoạn thẳng, 3 đoạn thẳng này tạo thành tam giác gì?
- Chia nhỏ bài toán thành các bài toán nhỏ
- Giải các bài toán nhỏ, sử dụng khái niệm hàm
- Ví dụ:
  - 3 đoạn thẳng có tạo thành tam giác?
  - 3 đoạn thẳng này có tạo thành tam giác đều?
  - 3 đoạn thẳng này có tạo thành tam giác cân?
  - 3 đoạn thẳng này có tạo thành tam vuông?
  - 3 đoạn thẳng này có tạo thành tam giác vuông cân?

Phân Loại Tam Giác – laTamGiac

- 3 đoạn thẳng tạo thành tam giác

$$? \ \text{laTamGiac}(a, b, c)$$

```
bool laTamGiac(double a, double b, double c)
{
    return ((a + b > c)
         && (b + c > a)
         && (c + a > b));
}

int main()
{
    if (laTamGiac( … , … , … )) … … …
    return 0;
}
```

Phân Loại Tam Giác – laTamGiacCan

- 3 đoạn thẳng tạo thành tam giác cân

$$? \ \text{laTamGiacCan}(a, b, c)$$

```
bool laTamGiacCan(double a, double b, double c)
{
    return ((a == b)
         || (b == c)
         || (c == a));
}

int main()
{
    if (laTamGiacCan( … , … , … )) … … …
    return 0;
}
```

Phân Loại Tam Giác – laTamGiacDeu

- 3 đoạn thẳng tạo thành tam giác đều

$$? \ \text{laTamGiacDeu}(a, b, c)$$

```
bool laTamGiacDeu(double a, double b, double c)
{
    return ((a == b)
         && (b == c)
         && (c == a));
}

int main()
{
    if (laTamGiacDeu( … , … , … )) … … …
    return 0;
}
```

Phân Loại Tam Giác – laTamGiacVuong

- 3 đoạn thẳng tạo thành tam giác vuông

$$? \ \text{laTamGiacVuong}(a, b, c)$$

```
bool laTamGiacVuong(double a, double b, double
c)
{
    return ((a*a == b*b + c*c)
         || (b*b == c*c + a*a)
         || (c*c == a*a + b*b));
}
int main()
{
    if (laTamGiacVuong( … , … , … )) … … …
    return 0;
}
```

Phân Loại Tam Giác – laTamGiacVuong

- 3 đoạn thẳng tạo thành tam giác vuông

$$? \ \text{laTamGiacVuong}(a, b, c)$$

```
bool laTamGiacVuong(double a, double b, double
c)
{
    double bpCanhA = a * a;
    double bpCanhB = b * b;
    double bpCanhC = c * c;

    return ((bpCanhA == bpCanhB + bpCanhC)
         || (bpCanhB == bpCanhC + bpCanhA)
         || (bpCanhC == bpCanhA + bpCanhB));
}
int main() { … … … return 0; }
```

Phân Loại Tam Giác – laTamGiacVuongCan

- 3 đoạn thẳng tạo thành tam giác vuông cân ? $\text{laTamGiacVuongCan}(a, b, c)$

```
bool laTamGiacVuongCan(double a, double b,
double c)
{
    return (laTamGiacVuong(a, b, c)
         && laTamGiacCan(a, b, c));
}

int main()
{
    if (laTamGiacVuongCan( … , … , … )) … … …
    return 0;
}
```

Phân Loại Tam Giác – laTamGiacThuong

- 3 đoạn thẳng tạo thành tam giác thường

$$? \ \text{laTamGiacThuong}(a, b, c)$$

```
bool laTamGiacThuong(double a,
                     double b,
                     double c)
{
    return (laTamGiac(a, b, c));
}

int main()
{
    if (laTamGiacThuong( … , … , … )) … … …
    return 0;
}
```

Phân Loại Tam Giác

```
bool laTamGiac( … ) { … }
bool laTamGiacDeu( … ) { … }
bool laTamGiacCan( … ) { … }
bool laTamGiacVuong( … ) { … }
bool laTamGiacVuongCan( … ) { … }
bool laTamGiacThuong( … ) { … }

int main()
{
    (laTamGiac(…)) ? … … … : … … … ;
    (laTamGiacDeu(…)) ? … … … : … … … ;
    (laTamGiacCan(…)) ? … … … : … … … ;
    (laTamGiacVuong(…)) ? … … … : … … … ;
    (laTamGiacVuongCan(…)) ? … … … : … … … ;
    return 0;
}
```

Phân Loại Tam Giác

```
bool laTamGiac(double a, double b, double c)
{
    return ((a + b > c)
         && (b + c > a)
         && (c + a > b));
}
int main()
{
    double canhA, canhB, canhC;
    cin >> canhA >> canhB >> canhC;
    if (laTamGiac( canhA , canhB , canhC ))
        cout << "la tam giac" << endl;
    else
        cout << "khong la tam giac" << endl;
    return 0;
}
```

Khái Niệm Chung

- Hàm là một khối các mệnh lệnh `{ … }`
- Hàm thực hiện thao tác được xác định trước
- Hàm phải được đặt tên (giống biến)
- Hàm có kiểu (giống biến)
- Hàm phải được định nghĩa trước khi sử dụng
  - Giống khởi tạo (gán) biến?
- Mệnh lệnh cuối cùng trong khối các mệnh lệnh `{ … }` là mệnh lệnh **`return`** `… ;`
  - Thực hiện mệnh lệnh **`return`** trả về kết quả và thoát khỏi hàm

Định Nghĩa Hàm

- Cú pháp:

```
Kiểu_Dữ_Liệu tên_hàm(danh_sách_biến)
{
        // thân hàm
        // danh sách các mệnh lệnh
}
```

  - Kiểu dữ liệu của hàm (**`Kiểu_Dữ_Liệu`**)
    - **`int`**, **`double`**, **`bool`**, **`char`**, …
    - Nếu hàm không trả về gì, kiểu của hàm là **`void`**
  - Tên hàm (**`tên_hàm`**)
  - Danh sách dữ liệu đầu vào (**`danh_sách_biến`**)
    - Danh sách tham số (tham số cũng có kiểu)

Định Nghĩa Hàm

- Sử dụng câu lệnh **`return`** trả về kết quả hàm
- Thực hiện câu lệnh **`return`** kết thúc các thao tác của hàm
  - Thông thường **`return`** là câu lệnh cuối cùng trong thân hàm (định nghĩa của hàm)
- Kiểu dữ liệu của giá trị trả về, phải giống với kiểu của hàm (**`Kiểu_Dữ_Liệu`**)

```
int main()
{
    … … …
    return 0;
}
```

Vị Trí Đặt Định Nghĩa Hàm

- Hàm phải được định nghĩa trước khi sử dụng
- Định nghĩa hàm thường được đặt trước (ở trên và ở ngoài) **`int main()`**
- Sau khi định nghĩa có thể sử dụng trong **`int main()`** và ở các hàm khác

```
int luyThua(int co_so, int so_mu) { … … … }

int main()
{
    … = luyThua( … , … );
    return 0;
}
```

Sử Dụng Hàm

- Sau khi được định nghĩa hàm có thể được sử dụng bất kỳ đâu miễn sao dữ liệu trả về của hàm phù hợp (về cú pháp và kiểu dữ liệu) với mục đích sử dụng
  - Không thể **`cin >> luyThua(a, b);`**

```
int luyThua(int co_so, int so_mu) { … … … }

int main()
{
    int a, b; cin >> a >> b;
    cout << luyThua(a, b);
    int ket_qua = luyThua(a, b);
    cout << ket_qua;
    return 0;
}
```

Sử Dụng Hàm

- Hàm phải được sử dụng như miêu tả trong định nghĩa
- Hàm **power** cần 2 tham số để tính
- Phải cung cấp cho hàm **power** cả 2 tham số
- Nếu không trình biên dịch báo lỗi (cú pháp)

```
int luyThua(int co_so, int so_mu) { … … … }

int main()
{
    int a = luyThua(2, 3);
    int b = luyThua(2);
    int c = luyThua(2, 3, 4);
    return 0;
}
```

Sử Dụng Hàm

- Khai báo hàm trước và định nghĩa hàm sau **`main()`**

```
// khai báo hàm
int luyThua(int co_so, int so_mu);

int main()
{
    int a = luyThua(2, 3);
    return 0;
}

// định nghĩa hàm
int luyThua(int co_so, int so_mu)
{
    … … …
}
```

Sử Dụng Khai Báo – Định Nghĩa Hàm

- Đặt định nghĩa hàm nào trước **A** hay **B**
- Lưu ý:
  - định nghĩa hàm **A** có sử dụng (gọi) hàm **B**
  - định nghĩa hàm **B** có sử dụng (gọi) hàm **A**

```
int A( … ) { … … …  B( … );  … … … }
int B( … ) { … … …  A( … );  … … … }

int main()
{
    … … …
    return 0;
}
```

Sử Dụng Khai Báo – Định Nghĩa Hàm

- C++ cho phép sử dụng hàm trước khi định nghĩa
  - nếu có khai báo hàm trước khi sử dụng
- Sau khi khai báo hàm, định nghĩa hàm có thể đặt bất kỳ ở đâu
- Nên sử dụng cách này

```
// khai báo hàm trước int main()
int A( … );
int B( … );

int main() { … … …  return 0; }

// định nghĩa hàm sau int main()
int A( … ) { … … …  B( … );  … … … }
int B( … ) { … … …  A( … );  … … … }
```

Chữ Ký Hàm – Nguyên Mẫu Hàm

- Bỏ đi phần `{ … }` của định nghĩa hàm
- Bỏ đi `;` của khai báo hàm
- Phần còn lại (kiểu trả về, tên, danh sách tham số) của hàm giống hệt nhau ở cả định nghĩa và khai báo

```
// khai báo hàm trước int main()
int A( … );
int B( … );

int main() { … … …  return 0; }

// định nghĩa hàm sau int main()
int A( … ) { … … …  B( … );  … … … }
int B( … ) { … … …  A( … );  … … … }
```

Hàm Không Có Tham Số

```
void square()
{
    for (int i = 3; i > 0; i--)
    {
        for (int j = 3; j > 0; j--) cout << "*";
        cout << endl;
    }
    cout << endl;
}

int main()
{
    int soLuong; cin >> soLuong;
    for (int i = soLuong; i > 0; i--) square();
    return 0;
}
```

Hàm Không Có Tham Số

```
void square(char c)
{
    for (int i = 3; i > 0; i--)
    {
        for (int j = 3; j > 0; j--) cout << c;
        cout << endl;
    }
    cout << endl;
}
int main()
{
    int soLuong; cin >> soLuong;
    char c; cin >> c;
    for (int i = soLuong; i > 0; i--) square(c);
    return 0;
}
```

Hàm Có Nhiều Tham Số

```
void square(char c, int n)
{
    for (int i = n; i > 0; i--) {
        for (int j = n; j > 0; j--) cout << c;
        cout << endl;
    }
    cout << endl;
}
int main()
{
    const int DEM = 3;
    int col; cin >> col;
    char c; cin >> c;
    for (int i = DEM; i > 0; i--) square(c,col);
    return 0;
}
```

Truyền Tham Số Cho Hàm
truyền giá trị

- Truyền giá trị:
  - Bản sao giá trị của biến được truyền vào hàm
  - Hàm xử lý dữ liệu trên bản sao này
  - Không làm thay đổi giá trị của biến truyền vào hàm
  - Dùng trong trường hợp tính toán
- Ví dụ:
  - `double pow(double, double)`
  - `double sqrt(double)`
  - `int luyThua(int, int)`
  - `bool chiaHet(int, int)`
  - `bool laTamGiac(double, double, double)`

Truyền Tham Số Cho Hàm
truyền giá trị

```
void thayDoiTruyenGiaTri(int v);

int main()
{
    int var = 5;
    thayDoiTruyenGiaTri(var);
    cout << "main: var=" << var << endl;
    return 0;
}

void thayDoiTruyenGiaTri(int v)
{
    v = v * 100;
    cout << "ham: v=" << v << endl;
}
```

```
ham: v=500
main: var=5
```

Truyền Tham Số Cho Hàm
truyền tham chiếu

- Truyền tham chiếu
  - Hàm xử lý dữ liệu của chính biến truyền vào hàm
  - Có thể thay đổi giá trị của biến
  - Dùng trong trường hợp cần chuyển dữ liệu sau khi đã xử lý ra ngoài hàm, cho các hàm khác sử dụng

Truyền Tham Số Cho Hàm
truyền tham chiếu

```
void thayDoiTruyenThamChieu(int & v);

int main()
{
    int var = 5;
    thayDoiTruyenThamChieu(var);
    cout << "main: var=" << var << endl;
    return 0;
}

void thayDoiTruyenThamChieu(int & v)
{
    v = v * 100;
    cout << "ham: v=" << v << endl;
}
```

```
ham: v=500
main: var=500
```

Truyền Tham Số Cho Hàm
truyền giá trị

- Chương trình chạy không đúng, giá trị nhập vào không được gán cho biến, các biến chưa được nhập dữ liệu
  - Do cách truyền tham số chưa đúng

```
void nhapSoND(int soND)
{
    do { cin >> soND; } while (soND <= 0);
    return ;
}
int main()
{
    int soThu1, soThu2;
    do { cin >> soThu1; } while (soThu1 <= 0);
    nhapSoND(soThu2);
    return 0;
}
```

Truyền Tham Số Cho Hàm
truyền giá trị

- Bản sao giá trị của biến **soThu1** được tạo ra và truyền
- Vào trong hàm **nhapSoND**, giá trị này được gán cho biến **soND**
- Thay đổi giá trị biến **soND** trong hàm **nhapSoND**,
- Kết thúc hàm **nhapSoND**, giá trị của biến **soND** thay đổi
- Nhưng giá trị biến **soThu1** không thay đổi

```
void nhapSoND(int soND) { // soND = soThu1
    do { cin >> soND; } while (soND <= 0);
}
int main() {
    int soThu1; nhapSoND(soThu1);
    return 0;
}
```

Truyền Tham Số Cho Hàm
truyền tham chiếu **&**

- Chính biến **soThu1** được truyền vào hàm **nhapSoND**
- Nhưng được lưu dưới một tên khác (**soND**) trong hàm **nhapSoND**
- Thay đổi giá trị biến **soND** trong hàm **nhapSoND**
- Sẽ làm thay đổi giá trị biến **soThu1** ở trong hàm **main**
- Do thực chất là một biến, với tên khác nhau, trong hàm **main** là **soThu1**, trong hàm **nhapSoND** là **soND**

```
void nhapSoND(int & soND) { // soND = &soThu1
    do { cin >> soND; } while (soND <= 0);
}
int main() {
    int soThu1; nhapSoND(soThu1);
    return 0;
}
```

Hàm – Tính Tổng

```
int tong(int soThu1, int soThu2)
{
    return soThu1 + soThu2;
}
void tong(int soThu1, int soThu2, int& tong2So)
{
    tong2So = soThu1 + soThu2;
}
int main()
{
    int a, b, tong2So;
    cin >> a >> b;
    tong(a, b, tong2So);
    cout << tong(a, b) << " " << tong2So;
    return 0;
}
```

Một Số Nguyên Tắc

- Tất cả các hàm trong C++ đều ngang cấp
  - Hàm không được khai báo lồng với nhau
  - Thứ tự khai báo không quan trọng
- Hàm có thể nhận và xử lý nhiều tham số hoặc không có tham số nào
- Hàm có thể trả về **một** (hoặc không) giá trị
- Phải truyền đầy đủ số lượng tham số và đúng kiểu khi gọi hàm, nếu không sẽ lỗi dịch
- Biến khai báo trong hàm nào chỉ có phạm vi trong hàm đó, không sử dụng được những biến này trong các hàm khác

Quy Tắc Phạm Vi

- Phạm vi được xác định bắt đầu từ { đến }
  - Được áp dụng ở hàm (định nghĩa hàm), các cấu trúc điều khiển (thân cấu trúc) và các khối lệnh { }
- Biến được khai báo trong phạm vi nào chỉ có thể được sử dụng trong phạm vi đó
- Kết thúc một phạm vi (}), tất cả các biến vừa được khai báo trong phạm vi đó sẽ bị xóa
- Có thể khai báo các biến cùng tên trong các phạm vi khác nhau
- Các biến này là biến cục bộ
  - Duy trì điều khiển trong nội bộ phạm vi

Quy Tắc Phạm Vi

- Phạm vi toàn cục
  - Biến được khai báo ở bên ngoài **`main`**, và ngoài tất cả các hàm
  - Biến này sẽ được hiểu và có thể dùng ở mọi nơi trong chương trình
  - Hạn chế dùng, dễ gây lỗi không đáng có khi thay đổi giá trị biến không cẩn thẩn
  - Thường khai báo hằng số có phạm vi toàn cục để mọi hàm đều hiểu

Tại Sao Sử Dụng Hàm

- Giải quyết một bài toán phức tạp
  - Chia nhỏ bài toán ban đầu thành bài toán nhỏ hơn, đơn lẻ, dễ giải quyết
  - Xử lý từng bài toán nhỏ lần lượt
  - Kết hợp lại để giải bài toán ban đầu
- Ngôn ngữ lập trình bậc cao
  - Cung cấp công cụ để phân tách và xử lý bài toán nhỏ (khối lắp ghép - mô-đun hóa chương trình)
- Lợi ích
  - Tái sử dụng mã, dễ quản lý
  - Tránh các đoạn mã giống nhau lặp đi lặp lại
  - Tìm lỗi, gỡ lỗi dễ dàng, nhanh chóng