Hàm & Kỹ thuật tổ chức chương trình

Nhập môn lập trình

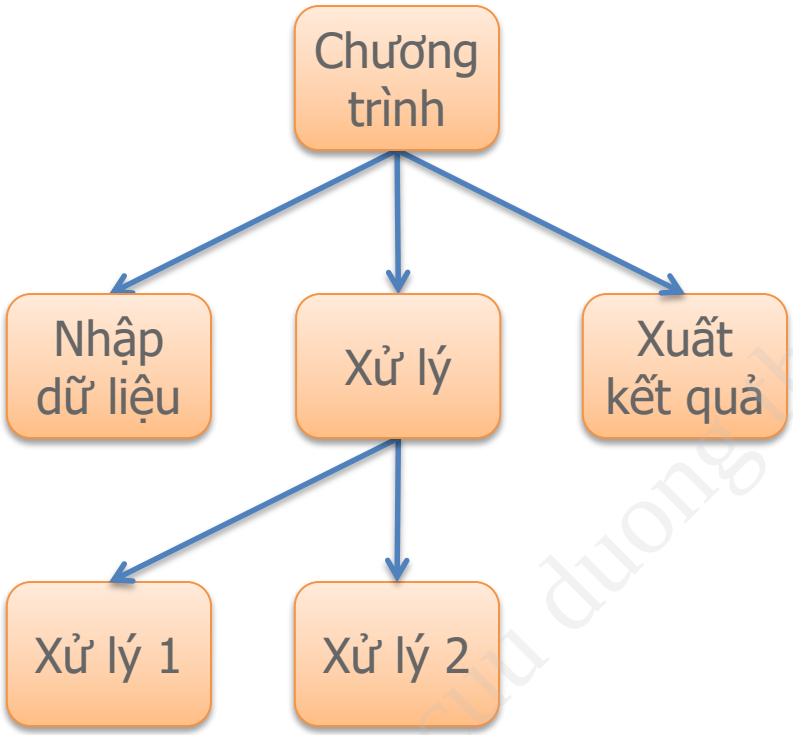
Trình bày: …; Email: …@fit.hcmus.edu.vn

Nội dung

- Giới thiệu
- Truyền tham số cho hàm
- Biến toàn cục và biến cục bộ
- Các ví dụ về ứng dụng hàm trong lập trình
- Hàm trong chương trình nhiều tập tin mã nguồn
- Các vấn đề tìm hiểu mở rộng kiến thức nghề nghiệp
- Thuật ngữ và bài đọc thêm tiếng Anh

Giới thiệu

Tiếp cận top-down



Tiếp cận top-down
trong lập trình cấu trúc

Chương trình lớn được chia thành các chương trình con nhỏ hơn nhằm dễ dàng phân chia và kiểm tra công việc hay sử dụng lại những bộ phận đã hoàn tất.

Đặc điểm

- Hàm có các đặc điểm sau:
  - Có một tên duy nhất.
  - Là một thành phần độc lập.
  - Thực hiện một công việc cụ thể.
  - Có thể nhận các đối số.
  - Có thể trả về giá trị cho chương trình gọi nó.

Nguyên mẫu hàm

return-type function_name(param-type param_name,
…, param-type param_name);

- Trong đó:
  - return-type: kiểu của giá trị hàm sẽ trả về, nếu không trả về gì cả thì kiểu trả về sẽ là void.
  - function_name: tên của hàm, thể hiện công việc hàm sẽ làm, nên bắt đầu bằng một động từ.
  - param-name, param-type: tên và kiểu tương ứng của tham số hình thức (formal parameter).
  - Được kết thúc bằng dấu chấm phẩy ;

Định nghĩa hàm

```
return-type function_name(param-type param_name,
                          ..., param-type param_name)
{
        // statements here…
}
```

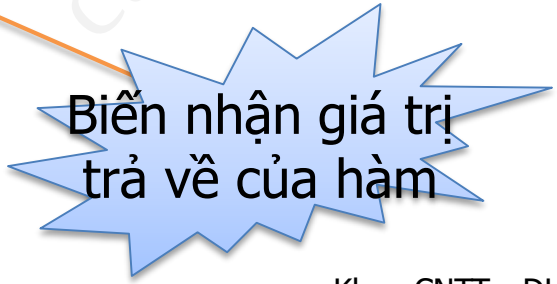
- Trong đó:
  - Dòng đầu là tiêu đề hàm (giống nguyên mẫu hàm nhưng không có ; và bắt buộc phải có tên tham số).
  - Tiếp theo là thân hàm (đặt trong {}) chứa các câu lệnh hàm sẽ thực hiện (phải có ít nhất một lệnh return nếu kiểu trả về không phải là void)

Phân biệt một cách tương đối

- Hàm có sẵn (trong ngôn ngữ hoặc do một hãng phần mềm viết để bán hoặc cho) như:
  - Hàm xuất, nhập thông tin: printf(), scanf(), …
  - Hàm toán học: sqrt(), pow(), abs(), sin(), …
- Hàm do người lập trình viết thêm như:
  - Hàm xuất, nhập thông tin: Nhập số dương, …
  - Hàm toán học: Tính căn bậc 3, tính căn bậc n, tính giai thừa, giải phương trình bậc 1, bậc 2, bậc 4 đối xứng, …

Ví dụ về hàm có sẵn

```
void main()
{
        int a = 7, b = 5;
        float z = 9;
        printf("a = %d\n", a);
        printf("b = ");
        scanf("%d", &b);
        z = (float)pow((double)b, (double)a);
}
```



Đối số

Ví dụ về hàm tự viết thêm

- Hàm tính $\sqrt[3]{x}$ ($x \in \mathbb{R}$) chưa có trong thư viện math.h

$$x \rightarrow \text{hàm } sqrt3 \rightarrow \sqrt[3]{x}$$

- Lưu ý:

$$\sqrt[3]{x} = \begin{cases} 0 & nếu\ x = 0 \\ pow(x, 1.0/3) & nếu\ x > 0 \\ -pow(-x, 1.0/3) & nếu\ x < 0 \end{cases}$$

- Khai báo hàm: double sqrt3(double x);

Định nghĩa hàm sqrt3()

```
double sqrt3(double x) {
        double y = 0;          // temporary variable
        if (x > 0)
                y = pow(x, 1/(double)3);
        else
                if (x < 0)
                        y = -pow(-x, 1/(double)3);
        return y;              // returns result
} // end of function
```

Ví dụ về hàm tự viết thêm

- Viết hàm tính $\sqrt[n]{x}$ $(x \in \mathbb{R}, n \in \mathbb{N}^*)$

  $x, n \rightarrow$ hàm $sqrtN \rightarrow \sqrt[n]{x}$ (nếu xác định)

- Lưu ý:
  - Nếu n lẻ thì $\sqrt[n]{x}$ luôn xác định.

$$\sqrt[n]{x} = \begin{cases} 0 & nếu\ x = 0 \\ pow(x, 1.0/n) & nếu\ x > 0 \\ -pow(-x, 1.0/n) & nếu\ x < 0 \end{cases}$$

  - Nếu n chẵn thì $\sqrt[n]{x}$ chỉ xác định khi $x \geq 0$.

$$\sqrt[n]{x} = pow(x, 1.0/n)$$

Khai báo hàm kèm ghi chú

```
// Function name    : sqrtN
// Description      : calculates n-th root of x
// Parameter        : double x
// Return type      : double
//       0          if n < 0
//       1          if n = 0
//       x^(1/n)    if n odd
//       x^(1/n)    if n even and x >= 0
//       0          if n even and x < 0
double sqrtN(double x);
```

Định nghĩa hàm sqrtN()

```
double sqrtN(double x)
{
        double y = 0;

        if (n <= 0 || (n % 2 ==0 && x < 0))
                return 0;
        if (n % 2 != 0)
        {
                if (x > 0)
                        y = pow(x, 1.0/n);
```

Định nghĩa hàm sqrtN()

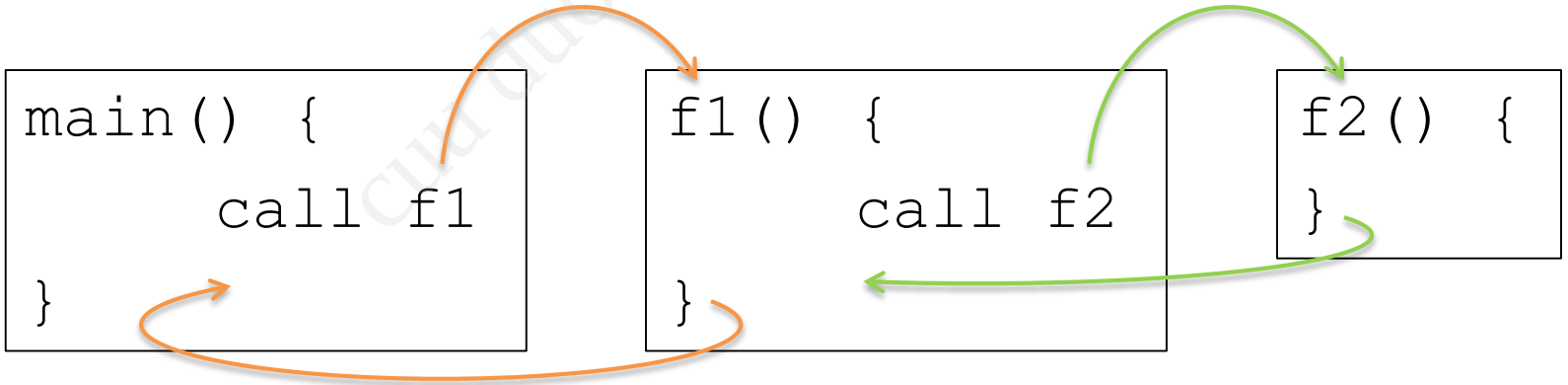
```
            else // n odd and x <= 0
                if (x < 0)
                    y = -pow(-x, 1.0/n);
        }
        else // n even and x > 0
            y = pow(x, 1.0/n);

        return y;
}
```

Truyền tham số cho hàm

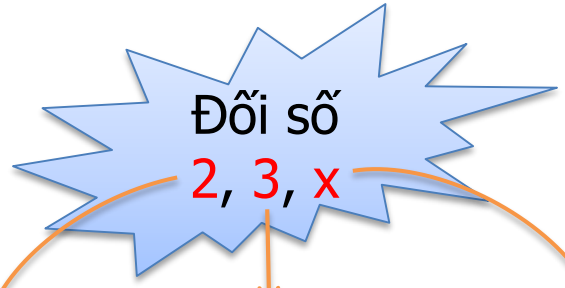
Sự thực thi của hàm

- Các câu lệnh bên trong hàm chỉ được thực thi khi hàm được gọi từ một phần khác của chương trình.
- Khi gọi hàm, chương trình có thể truyền đến hàm thông tin dưới dạng một hay nhiều đối số.



Khái niệm đối số

- Đối số (argument) hay tham số thực (actual parameter) là dữ liệu của chương trình truyền đến hàm có kiểu dữ liệu ứng với tham số hình thức được khai báo trong nguyên mẫu hàm. Dữ liệu này thường được hàm sử dụng để thực hiện công việc của nó.



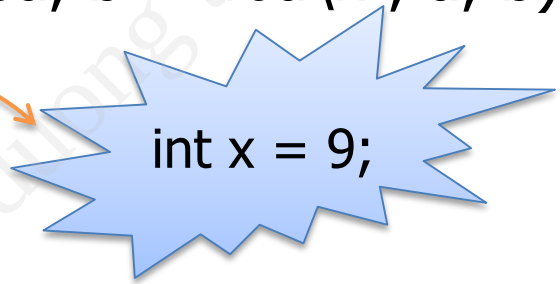

```
int SolveEq1(double a, double b, double &x);
```

Truyền đối số cho hàm

- Có hai cách truyền đối số
  - Truyền bằng giá trị (pass by value)
    - Đối số không đổi do hàm tạo bản sao của đối số khi nhận.
    - Thông thường là dữ liệu có sẵn.
    - Tham số hình thức tương ứng được gọi là tham trị.
  - Truyền bằng tham chiếu (pass by reference): C++
    - Đối số có thể thay đổi khi gọi hàm.
    - Thông thường là dữ liệu cần tính toán, xác định.
    - Tham số hình thức tương ứng được gọi là tham chiếu hay tham biến.

Ví dụ về tham trị

```
int Inc(int x);
void main() {
        int a = 9, b;
        b = Inc(a); // a is passed by value
        printf("a = %d, b = %d\n", a, b);
}
int Inc(int x) {
        x++;
        return x;
}
```



Ví dụ về tham biến

Địa chỉ của a

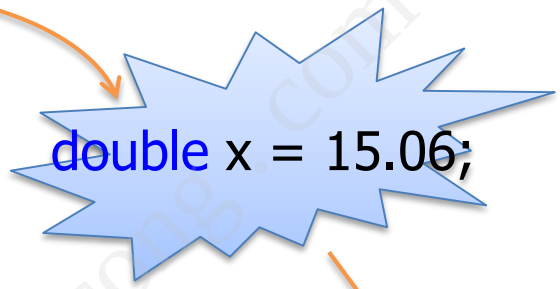
```
int Inc(int &x); // C++
void main() {
        int a = 9, b;
        b = Inc(a);
        printf("a = %d,
        b = %d\n", a, b);
}
int Inc(int &x) {
        x++;
        return x;
}
```

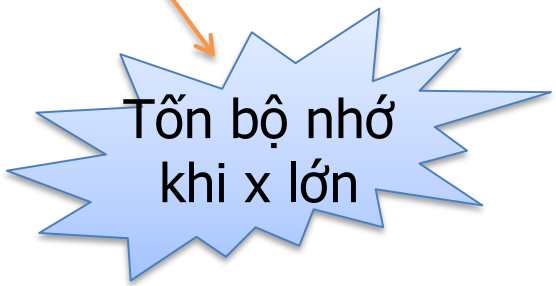
```
int Inc(int *x); /* C */
void main() {
        int a = 9, b;
        b = Inc(&a);
        printf("a = %d,
        b = %d\n", a, b);
}
int Inc(int *x) {
        (*x)++;
        return (*x);
}
```

int *x = &a;

Tham biến hằng

```
void f1(double x);
void f2(double &x);
void f3(const double &x);
void main() {
        double a = 15.06;
        f1(a); // passed by value
        f2(a); // passed by reference
        f3(a); // passed by const reference
}
// defines f1(), f2(), f3() here…
```





Lời gọi hàm

- Có hai cách để gọi hàm
  - Mọi hàm đều có thể được gọi bằng cách sử dụng tên hàm kèm danh sách các đối số trong một câu lệnh đơn. Nếu hàm có giá trị trả về, giá trị này sẽ bị bỏ qua.
  - Đối với các hàm có giá trị trả về, do các hàm này được quy thành một giá trị (do hàm trả về) nên chúng là các biểu thức C hợp lệ và có thể được sử dụng ở bất kỳ nơi đâu mà một biểu thức C có thể được sử dụng.

Ví dụ lời gọi hàm

```
void DoSomething();
int Sum(int x, int y);
void main() {
        DoSomething();
        Sum(1, 2); // the return value is discarded
        int x = Sum(1, 2);
        int y = Sum(1, Sum(2, 3));
        printf("%d\n", Sum(1, 2));
}
// defines DoSomething() and Sum() here…
```

Lưu ý về lời gọi hàm

- Nếu cố sử dụng các hàm có kiểu trả về là void như một biểu thức thì trình biên dịch sẽ phát sinh một thông báo lỗi.

```
void DoSomething();
void main() {
        DoSomething();
        int x = DoSomething(); // error
        printf("%d\n", DoSomething()); // error
}
// defines DoSomething() here…
```

Lưu ý về lời gọi hàm

- Hãy truyền đối số vào hàm để làm cho hàm tổng quát để có thể tái sử dụng.

```
int Sum() { // non generic
        int x, y;
        // inputs x, y here…
        return x + y;
}
int Sum(int x, int y) { // generic and thus reusable
        return x + y;
}
```

Lưu ý về lời gọi hàm

- Nên tận dụng ưu điểm của khả năng đặt hàm vào trong biểu thức nhưng tránh làm cho câu lệnh dài dòng, khó hiểu.

```
int Sum(int x, int y);
void main() {
        int a = 1, b = 2 , c = 3, x;
        printf("%d", Sum(a, Sum(b, c)));  // !!!
        x = Sum(b, c);
        printf("%d", Sum(a, x));          // better
}
```

Biến toàn cục và biến cục bộ

Khái niệm tầm vực của biến

- Là phạm vi hiệu quả của biến khi được khai báo trong chương trình
- Biến cục bộ (local variable)
  - Được khai báo bên trong hàm.
  - Chỉ có tác dụng trong hàm đó.
  - Được khởi tạo bởi một hằng số hoặc một biểu thức tương ứng với kiểu của biến.
  - Biến cục bộ sẽ bị xóa khỏi bộ nhớ ngay khi kết thúc hàm.

Khái niệm tầm vực của biến

- Biến toàn cục (global variable)
  - Được khai báo bên ngoài tất cả các hàm (kể cả hàm main()).
  - Có tác dụng trên toàn bộ chương trình(!).
  - Được khởi tạo một lần duy nhất bởi một hằng số tương ứng với kiểu của nó trước khi được sử dụng bên trong các hàm (tự động được gán giá trị 0 nếu không khởi gán tường minh).
  - Chỉ được giải phóng khi kết thúc chương trình.

Ví dụ biến toàn cục, cục bộ

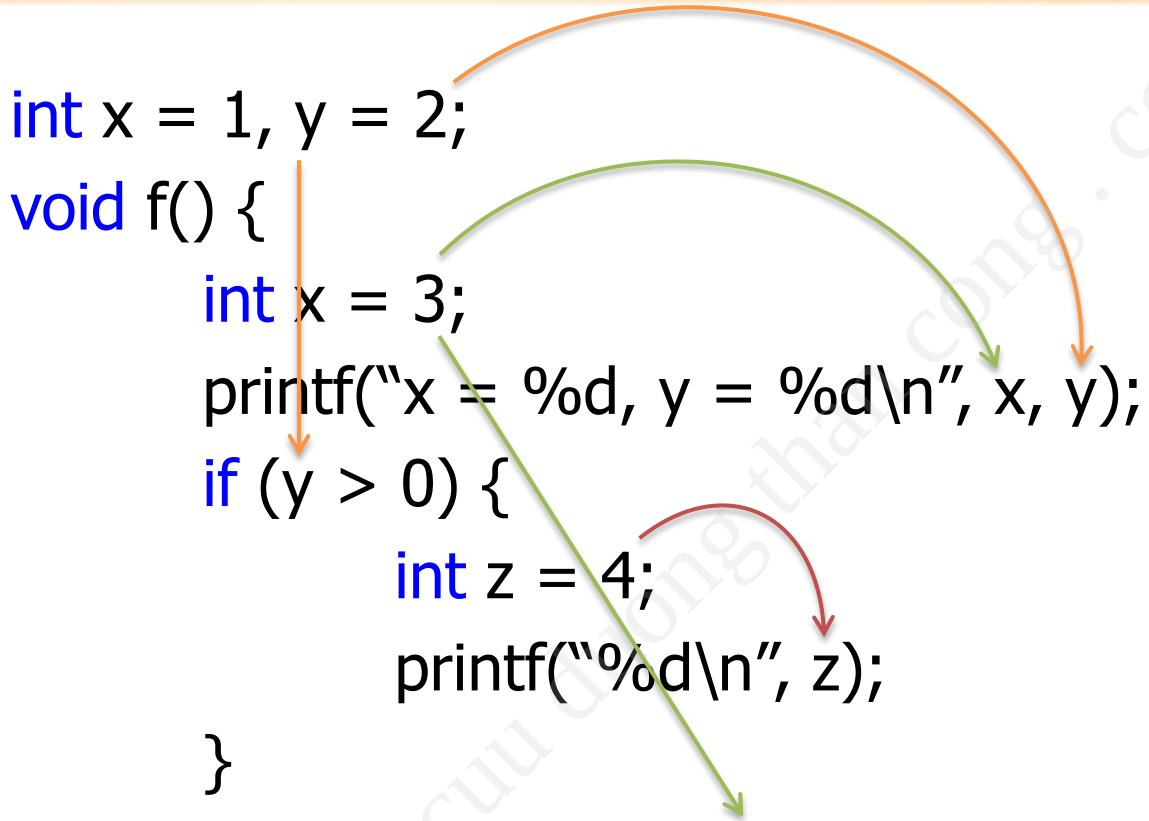
Biến toàn cục

```
int x = 999;
void f();
void main() {
        printf("%d\n", x);
        f();
}
void f() {
        printf("%d\n", x);
}
```

Biến cục bộ

```
void f();
void main() {
        int x = 999;
        printf("%d\n", x);
        f();
}
void f() {
        printf("%d\n", x);
}
```

Ví dụ biến toàn cục, cục bộ



```
int x = 1, y = 2;
void f() {
        int x = 3;
        printf("x = %d, y = %d\n", x, y);
        if (y > 0) {
                int z = 4;
                printf("%d\n", z);
        }
        printf("x = %d\n", x);
        printf("z = %d\n", z); // error
}
```

Nói thêm về biến toàn cục

- Biến toàn cục (global variable) là cách gọi khác của biến ngoài (external variable).
- Nói đúng ra, tầm vực của biến ngoài (hay biến toàn cục) là trong toàn bộ mã nguồn của tập tin chứa khai báo biến đó.
- Các chương trình C có kích thước không lớn chỉ được chứa trong một tập tin mã nguồn nên tầm vực là toàn bộ chương trình.
- Biến ngoài được khai báo tường minh bằng từ khóa extern.

Ví dụ khai báo biến ngoài

```
int x = 999; // external/global variable
void f();
void main() {
        extern int x;
        printf("%d\n", x);
        f();
}
void f() {
        extern int x;
        printf("%d\n", x);
}
```

Có thể bỏ từ khóa extern nếu trong cùng một tập tin mã nguồn

Sử dụng biến cục bộ

- Hạn chế sử dụng biến ngoài/toàn cục vì điều này phá vỡ tính độc lập đơn thể (modular independence), nguyên lý trung tâm của lập tình cấu trúc.
- Độc lập đơn thể là ý tưởng mỗi hàm hay đơn thể trong một chương trình chứa tất cả mã nguồn và dữ liệu cần thiết để thực hiện công việc của nó.
- Đối với các chương trình nhỏ thì việc sử dụng chung biến ngoài/toàn cục không quan trọng nhưng khi làm việc với các chương trình lớn hơn và phức tạp hơn thì sự quá ràng buộc vào biến ngoài sẽ nảy sinh nhiều vấn đề rắc rối.

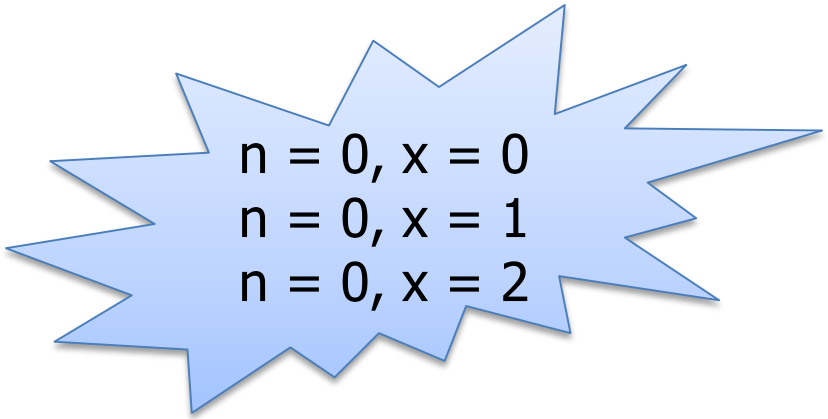
Khái niệm biến cục bộ tĩnh

- Mỗi khi chương trình thực thi lời khai báo biến cục bộ, một bản sao riêng biệt của biến cục bộ đó được tạo ra.
- Nếu biến cục bộ được khai báo là tĩnh (static) thì biến này sẽ được tạo ra một lần duy nhất ở lần đầu tiên khi chương trình thực thi lời khai báo của nó.
- Không như biến toàn cục, biến cục bộ tĩnh không bị truy cập và thay đổi bởi các hàm khác.

Ví dụ biến cục bộ tĩnh

```
void f() {
        static int n = 0;       // initialized once
        int x = 0;              // initialized n times
        printf("n = %d, x = %d\n", n++, x++);
}

void main() {
        int i;
        for (i = 0; i < 3; i++)
                f();
}
```



Dữ liệu nhập, xuất, trung gian

- Có 3 loại dữ liệu sau khi thực hiện yêu cầu gọi hàm:
  - Dữ liệu nhập: dữ liệu có sẵn, cần thiết để thực hiện hàm, thường được truyền ở dạng tham trị hoặc tham biến.
  - Dữ liệu xuất: dữ liệu hàm tính toán được, thường được trả về bằng lệnh return hoặc ở dạng tham biến.
  - Dữ liệu trung gian: dữ liệu do hàm tạo ra trong quá trình thực hiện công việc, thường phục vụ cho việc tính toán dữ liệu xuất.

Ví dụ các loại dữ liệu

```
// returns f(x, y) = ax + by and reverses the signs of a, b if f < 0
int Calculate(float &a, float &b, float x, float y) {
        int temp1, temp2, f;
        temp1 = a * x;
        temp2 = b * y;
        f = temp1 + temp2;
        if (f < 0) {
                a = -a;
                b = -b;
        }
        return f;
}
```

Các ví dụ về ứng dụng hàm trong lập trình

Ví dụ 1: Hàm giải PT bậc 1

- Viết chương trình giải phương trình bậc 1:

$$ax + b = 0 \ (a, b \in \mathbb{R})$$

  - Cách 1: Viết trực tiếp ngay trong hàm main() (nhập a, b rồi xét từng trường hợp để in ra kết quả). Cách này không thể dùng lại sau này khi cần để giải phương trình bậc nhất.
  - Cách 2: Viết một hàm nhiệm vụ giải phương trình bậc 1, hàm được sử dụng lại trong chương trình chính.

Khai báo hàm SolveEq1()

$$a, b \rightarrow \text{hàm } SolveEq1 \rightarrow x, nSol$$

- Khai báo hàm:

```
int SolveEq1(float a, float b, float &x);
```

- Lưu ý: số nghiệm nSol không thấy trong khai báo hàm sẽ được tính toán và ghi vào biến tạm rồi trả về bởi lệnh return.
- Định nghĩa các hằng số đặc biệt:

```
#define VODINH -1
```

Định nghĩa hàm SolveEq1()

```
int SolveEq1(float a, float b, float &x) {
        int nSol = 0;
        if (a != 0) {
                x = -b/a;
                nSol = 1;
        }
        else
                if (b == 0)
                        nSol = VODINH;
        return nSol;
}
```

Sử dụng hàm SolveEq1()

```
void main() {
        float a, b, x;

        // inputs a, b here…

        int nSol = SolveEq1(a, b, x);
        switch (nSol) {
                // checks nSol here…
        }
}
```

Ví dụ 2: Hàm giải PT bậc 2

- Viết hàm giải phương trình bậc 2:

$$ax^2 + bx + c = 0$$

$$a, b, c \rightarrow \text{hàm } SolveEq2 \rightarrow x_1, x_2, nSol$$

- Khai báo hàm:

```
int SolveEq2(float a, float b, float c,  float &x1, float &x2);
```

Định nghĩa hàm SolveEq2()

```
int SolveEq2(float a, float b, float c, float &x1, float &x2) {
        int nSol = 0;
        float delta;

        if (a == 0)
                return SolveEq1(a, b, x1); // reuses SolveEq1()

        delta = b*b – 4*a*c;
        if (delta < 0)
                return 0;
```

Định nghĩa hàm SolveEq2()

```
    if (delta == 0) {
            x1 = x2 = -b/(2*a);
            nSol = 1;
    }
    else { // delta > 0
            x1 = (-b – sqrt(delta))/(2*a);
            x2 = (-b + sqrt(delta))/(2*a);
            nSol = 2;
    }
    return nSol;
}
```

Sử dụng hàm SolveEq2()

```
void main() {
        float a, b, c, x1, x2;

        // inputs a, b, c here…

        int nSol = SolveEq2(a, b, c, x1, x2);
        switch (nSol) {
                // checks nSol here…
        }
}
```

Ví dụ 3. Giải PT đối xứng bậc 4

- Giải phương trình:

$$ax^4 + bx^3 + cx^2 + bx + a = 0$$

$$a, b, c \rightarrow \text{hàm } SolveEq4Sym \rightarrow x_1, x_2, x_3, x_4, nSol$$

- Khai báo hàm:

```
int SolveEq4Sym(float a, float b, float c,
                float &x1, float &x2, float &x3, float &x4);
```

Phác thảo cách giải

- Nếu $a = 0$ thì phương trình thành:

$$x(bx^2 + cx + b) = 0$$

- Do đó:

$$\left[\begin{array}{c} x = 0 \\ bx^2 + cx + b = 0 \end{array}\right.$$

- Dùng lại hàm SolveEq2() đã viết để giải phương trình bậc 2: $bx^2 + cx + b = 0$

Phác thảo cách giải

- Nếu $a \neq 0$ thì do $x = 0$ không phải là nghiệm nên ta có biến đổi tương đương:

$$-a\left(x^2 + \frac{1}{x^2}\right) + b\left(x + \frac{1}{x}\right) + c = 0$$

- Đặt $Y = x + \frac{1}{x} \Rightarrow Y^2 = x^2 + \frac{1}{x^2} + 2$
- Phương trình trở thành:

$$a(Y^2 - 2) + bY + c = 0$$
$$\Leftrightarrow aY^2 + bY + (c - 2a) = 0$$

Phác thảo cách giải

- Dùng hàm SolveEq2() giải phương trình $aY^2 + bY + (c - 2a) = 0$ để tìm nghiệm Y.
- Để tìm x, sử dụng hàm SolveEq2()
  - TH1. Vô nghiệm $\Rightarrow$ Phương trình ban đầu vô nghiệm.
  - TH2. Có 1 nghiệm Y_1: gọi SolveEq2() để giải:
    $$x^2 - Y_1 x + 1 = 0$$
  - TH3. Có 2 nghiệm Y_1, Y_2: tương tự TH2.

Định nghĩa hàm SolveEq4Sym()

```
int SolveEq4Sym(float a, float b, float c,
                float &x1, float &x2, float &x3, float &x4)
{
    int nSol;
    if (a == 0) {
        x1 = 0;
        nSol = SolveEq2(b, c, b, x2, x3);
        if (nSol != VODINH)
            nSol++;
    }
```

Định nghĩa hàm SolveEq4Sym()

```
else {
        float Y1, Y2;
        int nSol1 = SolveEq2(a, b, c-2*a, Y1, Y2);
        switch (nSol1) {
        case 0:
                nSol = 0;
                break;
        case 1:
                nSol = SolveEq2(1, -Y1, 1, x1, x2);
                break;
```

Định nghĩa hàm SolveEq4Sym()

```
case 2:
        int nSol2 = SolveEq2(1, -Y1, 1, x1, x2);
        switch (nSol2) {
        case 0:
                nSol = SolveEq2(1, -Y2, 1,
                                        x1, x2);
                break;
        case 1:
                nSol = SolveEq2(1, -Y2, 1,
                                        x2, x3);
                break;
```

Định nghĩa hàm SolveEq4Sym()

```
                case 2:
                        nSol = SolveEq2(1, -Y2, 1,
                                                x3, x4);
                        break;
                }
                break;
            }
        }
        return nSol;
}
```

Hàm trong chương trình nhiều tập tin mã nguồn

Lập trình đơn thể

- Chương trình với một tập tin mã nguồn chỉ phù hợp với các chương trình nhỏ.
- Khi đặt một tập các hàm có mục đích tổng quát vào một tập tin riêng, ta có thể sử dụng lại các hàm này ở các chương trình khác.
- Khi viết chương trình gồm nhiều tập tin mã nguồn, mỗi tập tin mã nguồn được gọi là một đơn thể (module). Cách lập trình như vậy gọi là lập trình đơn thể (modular programming), có liên quan rất gần với lập trình cấu trúc.

Tổ chức mã nguồn nhiều tập tin

- Mỗi chương trình C chỉ có duy nhất một hàm main().
- Đơn thể chứa hàm main() được gọi là đơn thể chính, các đơn thể khác được gọi là đơn thể phụ.
- Một tập tin tiêu đề riêng rẽ thường được đi kèm với mỗi đơn thể phụ.

mymath.h và mymath.c

```
/* mymath.h: header file for mymath.c */
double sqrt3(double x);
double sqrtN(double x);
/* end of mymath.h */
```

```
/* mymath.c: module containing math functions */
#include "mymath.h"
double sqrt3(double x) { /* statements */ }
double sqrtN(double x) { /* statements */ }
/* end of mymath.c */
```

sample.c (đơn thể chính)

```
#include <stdio.h>
#include "mymath.h"
void main() {
        int x;
        printf("Enter an integer value: ");
        scanf("%d", &x);
        printf("The 3rd root of %d is %.lf\n",
                                        x, sqrt3((double)x);
        /* other statements here... */
}
```

Phạm vi của hàm và biến toàn cục

- Hàm và biến toàn cục (hay biến ngoài) không tự động được thấy trong các đơn thể khác.
- Khai báo để các đơn thể khác có thể thấy được hàm hay biến toàn cục trong các đơn thể khác:
  - Hàm: sử dụng chỉ thị #include (ví dụ trước)
  - Biến toàn cục: sử dụng từ khóa extern

Ví dụ khai báo biến toàn cục

```
/* main module: sample.c */
int x = 99, y; /* the compiler automatically initializes y to 0 */
void main() {/* statements */ }
```

```
/* secondary module: mod1.c */
extern int x, y;
void func1() { /* statements */ }
```

```
/* secondary module: mod2.c */
extern int x;
void func2() { /* statements */ }
```

Các vấn đề tìm hiểu mở rộng kiến thức nghề nghiệp

Hàm trùng tên

- Nhu cầu
  - Thực hiện một công việc với nhiều cách khác nhau. Nếu các hàm khác tên sẽ khó nhớ.
- Ví dụ:
  - Các hàm tính trị tuyệt đối trong C (math.h)
    - int abs(int n);
    - long labs(long n);
    - double fabs(double n);
  - Các hàm tính căn bậc 2: sqrt(), sqrtf()

Hàm trùng tên

- Khái niệm
  - Là các hàm cùng tên nhưng có tham số đầu vào hoặc kiểu trả về khác nhau nhằm cho phép người dùng chọn cách thuận lợi nhất để thực hiện công việc.
  - Nguyên mẫu hàm khi bỏ tên tham số phải khác nhau.
  - Việc sử dụng các hàm trùng tên được gọi là nạp chồng hay quá tải (overload) hàm.

Ví dụ hàm trùng tên

```
// prints integers from 1 to n
void PrintIntegers(int n);

// prints integers from x to y
void PrintIntegers(int x, int y);

// prints integers from x to y
// with an arithmetic progression a
void PrintIntegers(int x, int y, int a);
```

Chú ý về hàm trùng tên

- Các hàm sau đây là như nhau do cùng nguyên mẫu hàm: int Sum(int, int);

```
// calculates a + b
int Sum(int a, int b);

// calculates b + a
int Sum(int b, int a);

// calculates x + y
int Sum(int x, int y);
```

Sự nhập nhằng, mơ hồ

```
float f(float x)        { return x/2; }
double f(double x)      { return x/2; }

void main() {
        float x = 29.12;
        double y = 17.06;
        printf("%.2f\n", f(x));                 // float
        printf("%.2lf\n", f(y));                // double
        printf("%.2f\n", f(10));                // ???
        printf("%.2f\n", f((float)10));         // float
}
```

Sự nhập nhằng, mơ hồ

```
void f(unsigned char c) { printf("%d", c); }
void f(char c) { printf("%c", c); }

void main()
{
        f('A');                 // char
        f(65);                  // ???
        f((char)65);            // char
        f((unsigned char)65);   // unsigned char
}
```

Sự nhập nhằng, mơ hồ

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

void main()
{
        int x = 1, y = 2;
        printf("%d", f(x, 2));          // b = 2
        printf("%d", f(x, y));          // ???
}
```

Sự nhập nhằng, mơ hồ

```
int f(int a) { return a*a; }
int f(int a, int b = 1) { return a*b; }

void main()
{
        printf("%d\n", f(2912, 1706));
        printf("%d\n", f(2912));            // ???
}
```

Hàm có đối số mặc định

- Khái niệm
  - Là hàm có một hay nhiều tham số hình thức được gán sẵn giá trị mặc định. Các tham số này nhận giá trị mặc định đó nếu không có đối số tương ứng được truyền vào.
  - Các tham số mặc định phải được dồn về tận cùng bên phải.
- Ví dụ

```
void PrintFraction(int num, int denom = 1);
```

Hàm có đối số mặc định

- Lưu ý:
  - Muốn truyền đối số khác thay cho đối số mặc định, phải truyền đối số thay cho các đối số mặc định trước nó.
- Ví dụ:

```
void SolveEq2(int a, int b = 0, int c = 0);
```

- Giải phương trình: $2x^2 + 0x + 3 = 0$
  - Sai: `SolveEq2(2, 3);` `// a=2, b=3, c=0`
  - Đúng: `SolveEq2(2, 0, 3);` `// a=2, b=0, c=3`

Hàm có đối số mặc định

- Lưu ý:
  - Nếu $x = a$ thường xuyên xảy ra thì nên chuyển x thành tham số có đối số mặc định là a.

    Ví dụ, $Gender = 0$ (Nam), $Age = 18$.
  - Nếu $x = a$ và $y = b$ thường xuyên xảy ra nhưng $y = b$ thường xuyên hơn thì nên đặt tham số mặc định y sau x.

    Ví dụ, trong cùng lớp học $Age = 18$ xảy ra nhiều hơn $Gender = 0$ do đó nên đặt Age sau $Gender$.

Hàm có tham số là hàm

- Khái niệm
  - Hàm có thể truyền vào hàm khác dưới dạng đối số đầu vào.
  - Việc khai báo tham số là hàm tương tự như khai báo nguyên mẫu hàm (không cần tên các tham số hình thức)
  - Chỉ được phép truyền các hàm có nguyên mẫu hàm (sau khi bỏ đi tên các tham số hình thức) giống với nguyên mẫu hàm của tham số hình thức hàm được khai báo.

Ví dụ hàm có tham số là hàm

```
int FindBestNumber(int a, int b, int Better(int, int)) {
        int numBest = a;
        if (Better(b, a)) {
                numBest = b;
        }
        return numBest;
}
int MaxNumber(int x, int y) { return x > y; }
int MinNumber(int x, int y)  { return x < y; }
```

Ví dụ hàm có tham số là hàm

```
int FindBestNumber(int a[], int n, int Better(int, int)) {
        int i, idBest = 0;
        for (i = 1; i < n; i++) {
                if (Better(a[i], a[idBest]))
                        idBest = id;
        }
        return a[idBest];
}
int MaxNumber(int x, int y) { return x > y; }
int MinNumber(int x, int y)  { return x < y; }
```

Hàm đệ qui

- Khái niệm
  - Đệ qui chỉ một tình huống mà trong đó hàm gọi chính nó theo cách trực tiếp hay gián tiếp.
- Ví dụ
  - Tính giai thừa: $n! = n * (n-1) * \cdots * 2 * 1$
  - Do $(n-1) * \cdots * 2 * 1 = (n-1)! \Rightarrow n! = n * (n-1)$
  - Tương tự $(n-1)! = (n-)! * (n-1)$
  - Tiếp tục cho đến khi tính 1! ta có ngay kết quả là 1, thế ngược lại tính được $n!$

Ví dụ hàm đệ qui

- Khai báo hàm:

```
unsigned int factorial(unsigned int n);
```

- Định nghĩa hàm:

```
unsigned int factorial(unsigned int n)
{
        if (n == 1)
                return 1;
        else
                return n * factorial(n – 1);
}
```

So sánh với các NNLT khác

Tiêu chí so sánh/Ngôn ngữ	C	C++	C#	Java
Khai báo hàm độc lập với các thành phần khác	✓	✓		
Khai báo hàm (phương thức) trong lớp đối tượng (class)		✓	✓	✓
Truyền bằng giá trị (tham trị)	✓	✓	✓	✓ (kiểu cơ sở)
Truyền bằng địa chỉ	✓	✓		
Truyền bằng biến (tham biến/tham chiếu)		✓ (sử dụng &)	✓ (sử dụng từ khóa ref, out)	✓ (đối tượng và mảng)
Tham số có giá trị mặc định		✓	✓	✓
Hàm trùng tên (nạp chồng hàm)		✓	✓	✓

Thuật ngữ và bài đọc thêm tiếng Anh

Thuật ngữ tiếng Anh

- ***function***: hàm (chương trình con)
- ***structured programming***: lập trình cấu trúc
- ***modular programming***: lập trình đơn thể
- ***parameter***: tham số
- ***argument*** : đối số
- ***formal parameter***: tham số hình thức, tương đương với ***parameter***
- ***actual parameter***: tham số thực, tương đương với ***argument***
- ***function prototype***: nguyên mẫu hàm
- ***function header***: tiêu đề hàm
- ***function declaration***: khai báo hàm
- ***function definition***: định nghĩa hàm

Thuật ngữ tiếng Anh

- ***local variable***: biến cục bộ
- ***extern varialbe***: biến ngoài
- ***global variable***: biến toàn cục, tương tự ***extern variable***
- ***call by value***: truyền đối số bằng giá trị (tham trị)
- ***call by reference***: truyền đối số bằng tham biến (tham chiếu)
- ***scope***: tầm vực, phạm vi hiệu quả
- ***recursion***: sự đệ qui
- ***overload***: nạp chồng, quá tải
- ***ambiguity***: nhập nhằng, mơ hồ

Bài đọc thêm tiếng Anh

- **Bradley L. Jones and Peter Aitken**, *Teach Yourself C in 21 days*, 6th Edition, SAMS, 2003.
  - Day 5. Packaging Code in Functions, pp. 97-122.
  - Day 12. Understanding Variable Scope, pp. 285-303.
  - Day 21. Advanced Compiler Use – Programming with Multiple Source-Code Files, pp. 593-600.
- **Bjarne Stroustrup**, *The C++ Programming Language*, 3rd Edition, AT&T, 1997.
  - Chapter 7. Functions, pp. 143-164.
  - Chapter 9. Source Files and Programs, pp. 197-220.