

# Ngôn ngữ lập trình C

## Bài 6. Hàm

*Soạn bởi: TS. Nguyễn Bá Ngọc*

# Các hàm đã sử dụng

*Chúng ta đã sử dụng hàm ngay từ chương trình đầu tiên:*

- Hàm **main**:
  - Mỗi chương trình phải có đúng 1 hàm **main**.
  - Được gọi theo quy ước khi bắt đầu thực hiện chương trình.
  - Không được khai báo **inline** hoặc **static**.
- Các hàm trong thư viện chuẩn:
  - **printf** - Hàm xuất dữ liệu theo định dạng.
  - **scanf** - Hàm nhập dữ liệu theo định dạng.
  - **sqrt** - Hàm lấy căn bậc 2.
  - ...

*Bài giảng này sẽ tiếp tục cung cấp các chi tiết về định nghĩa hàm, khai báo hàm và gọi hàm.*

## Ví dụ 6.1. Tính Ư'SCLN

*Cho chương trình tính Ư'SCLN của 2 số nguyên a và b*

```
1  #include <stdio.h>
2  int main() {
3      int a, b;
4      printf("Nhập a b: ");
5      scanf("%d%d", &a, &b);
6      while (b != 0) {
7          int tmp = a % b;
8          a = b;
9          b = tmp;
10     }
11     printf("USCLN = %d\n", a);
12 }
```

Nhập a b: 18 30 USCLN = 6
------------------------------

*Hãy điều chỉnh chương trình để tính Ư'SCLN của 3 số nguyên a, b, c?*

## Ví dụ 6.1. Tính ƯSCLN<sub>(2)</sub>

Sử dụng 1 vòng lặp while để tính ƯSCLN(a, b) và 1 vòng lặp while tương tự để tính ƯSCLN(ƯSCLN(a, b), c)? **X**

```
1  #include <stdio.h>
2  int uscln(int a, int b) {
3      while (b != 0) {
4          int tmp = a % b;
5          a = b;
6          b = tmp;
7      }
8      return a;
9  }
10 int main() {
11     int a, b, c;
12     printf("Nhập a, b, c: ");
13     scanf("%d%d%d", &a, &b, &c);
14     printf("ƯSCLN(%d, %d, %d) = %d\n",
15           a, b, c,
16           uscln(uscln(a, b), c));
17 }
```

Sử dụng hàm để tính ƯSCLN? **✓**

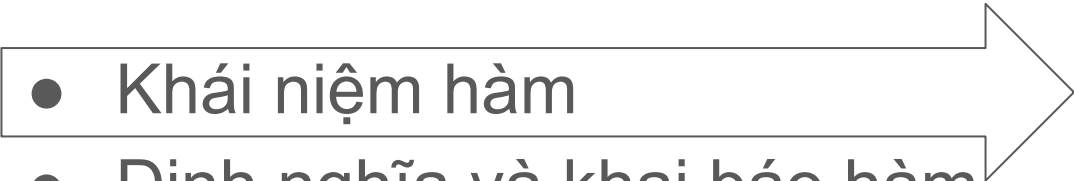
Nhập a, b, c: 18 30 9  
ƯSCLN(18, 30, 9) = 3

*Thay vì lặp nhiều lần 1 đoạn mã nguồn, với hàm chúng ta có thể sử dụng 1 đoạn mã nguồn nhiều lần.*

# Nội dung

- Khái niệm hàm
- Định nghĩa và khai báo hàm
- Biểu thức gọi hàm
- Biểu thức toán học

# Nội dung

- 
- Khái niệm hàm
  - Định nghĩa và khai báo hàm
  - Biểu thức gọi hàm
  - Biểu thức toán học

# Khái niệm hàm

- Hàm là tính năng lập trình cho phép đóng gói 1 đoạn mã nguồn để có thể tái sử dụng nhiều lần mà không cần lặp lại mã nguồn.
- Ở mức độ nhất định, tương tự hàm trong toán học, hàm trong C cũng có thể nhận dữ liệu đầu vào qua các tham số và trả về 1 giá trị đầu ra, tuy nhiên không bắt buộc.
- Khác hàm trong toán học, hàm trong C cũng có thể không nhận dữ liệu đầu vào và/hoặc không trả về kết quả, và có thể có các hiệu ứng.

# Khái niệm hàm<sub>(2)</sub>

- Hàm ẩn các chi tiết triển khai, người lập trình có thể sử dụng hàm như thao tác bậc cao.
- Với hàm 1 vấn đề tính toán lớn có thể được chia nhỏ và xử lý từng phần bởi nhiều người.

Hàm là tính năng quan trọng của bất kỳ NNLT nào.



# Kiểu hàm

- Kiểu hàm có thể được hiểu là tập hợp của các hàm có cùng kiểu trả về và cùng danh sách kiểu của tham số.
- Kiểu hàm còn được coi là kiểu suy diễn từ kiểu của giá trị mà nó trả về.
  - `int f(int a, int b);` // hàm trả về int ...
- Trong thực tế thường chỉ sử dụng kiểu con trỏ hàm
  - `int (*)(int, int)` là kiểu con trỏ hàm nhận 2 tham số kiểu int và trả về giá trị kiểu int
  - *(Nội dung chi tiết về con trỏ hàm sẽ được cung cấp sau)*

# Nội dung

- Khái niệm hàm
- Định nghĩa và khai báo hàm
- Biểu thức gọi hàm
- Biểu thức toán học

# Tổng quan về khai báo và định nghĩa hàm

*NNLT C tách rời 2 khái niệm khai báo hàm và định nghĩa hàm.*

- Khai báo hàm cho biết kiểu trả về của hàm và có thể cho biết kiểu của các tham số nhưng không bao gồm thân hàm.
- Định nghĩa hàm bao gồm cả khai báo hàm với kiểu của các tham số và thân hàm.
- Hàm phải được khai báo trước khi sử dụng/xuất hiện trước câu lệnh gọi hàm, để sinh mã gọi hàm trình biên dịch chỉ sử dụng khai báo hàm.
- Hàm có thể được định nghĩa trong cùng đơn vị biên dịch mà nó được sử dụng hoặc trong 1 đơn vị biên dịch khác
- Trường hợp chỉ có khai báo của hàm được sử dụng nhưng không có định nghĩa hàm/trình biên dịch không tìm thấy định nghĩa hàm, sẽ làm phát sinh lỗi ở pha lỗi ghép nối.

# Tổng quan khai báo và định nghĩa hàm<sub>(2)</sub>

- Phải xác định được 1 định nghĩa duy nhất của hàm được gọi cho mỗi biểu thức gọi hàm.
- Theo quy chuẩn ISO hàm được khai báo và định nghĩa trong phạm vi tệp.

*(Mở rộng GNU còn cho phép định nghĩa hàm trong hàm)*

# Tổ chức mã nguồn

- Thông thường các khai báo hàm được đặt trong các tệp .h và được chèn vào các đơn vị biên dịch để sử dụng.
  - Giảm trùng lặp mã nguồn và hỗ trợ quản lý mã nguồn.
  - Đảm bảo tính nhất quán trong nhiều đơn vị biên dịch.
- Mỗi hàm thường được định nghĩa 1 lần, các định nghĩa hàm thường được đặt trong tệp .c.
  - Định nghĩa hàm cũng có thể được cung cấp ở dạng mã máy / đã được biên dịch, điển hình như các hàm thư viện chuẩn.
- *(Tạm thời chúng ta chưa phân tích chi tiết về hàm **inline** và hàm **static**)*
  - Hàm **inline**: Trình biên dịch có thể thay thế câu lệnh gọi hàm bằng mã nguồn dựa trên thân hàm ở vị trí đó.
  - Hàm **static**: Hàm riêng trong phạm vi đơn vị biên dịch.

# Khuôn mẫu định nghĩa hàm

*Hàm có thể được định nghĩa theo cấu trúc:*

*kiểu-trả-về* **tên-hàm**(*mô-tả-tham-số<sub>opt</sub>*) **câu-lệnh-gộp**

- *kiểu-trả-về*: Kiểu dữ liệu của giá trị được trả về bởi hàm
  - Nếu hàm không trả về giá trị thì kiểu trả về được để là **void**.
  - Nếu để trống thì kiểu trả về được mặc định là int (*tính năng cũ có thể không còn được hỗ trợ trong tương lai*).
- **tên-hàm**: Là định danh có thể được sử dụng để gọi hàm.
- *mô-tả-tham-số<sub>opt</sub>*: Hàm có thể không có tham số, hoặc có danh sách tham số cố định, hoặc có danh sách tham số thay đổi / không định trước. Các tham số (nếu có) có phạm vi khối và có thể được sử dụng trong thân hàm.
- **câu-lệnh-gộp** - Thân hàm, được thực hiện khi hàm được gọi và sau khi giá trị của các tham số đã được thiết lập.

# Danh sách tham số rỗng

- Trong định nghĩa hàm chúng ta có thể để trống danh sách tham số hoặc sử dụng từ khóa **void** (có cùng ý nghĩa) nếu hàm không có tham số.
- Trong khai báo hàm từ khóa **void** ở vị trí mô tả tham số có nghĩa là hàm không có tham số.
- Để trống danh sách tham số () có nhiều ý nghĩa khác nhau phụ thuộc vào ngữ cảnh:
  - Trong định nghĩa: Để trống = Không có tham số
  - Trong khai báo: Để trống = Chưa biết số lượng tham số (!= không có tham số).
    - `void foo(void);` // Khai báo foo là hàm không có tham số
    - `void foo2();` // - chưa biết về tham số của foo2 (có thể có hoặc không)

# Danh sách tham số cố định

- Định dạng danh sách kiểu / khai báo từng tham số hiện đang được sử dụng phổ biến:
  - `void f(int a, int b, int c) { } // OK`
  - `void f(int a, b, c) { } // Lỗi cú pháp: cần khai báo từng tham số`
  - Khi gọi hàm các đối số được ép kiểu của tham số.
  - Phong cách hiện đại.
  - *(Tiếp theo chúng ta sẽ chỉ sử dụng định dạng này)*
- Các tham số cũng có thể được khai báo theo hình thức danh sách định danh:
  - `void f(a, b, c) { } // a, b, c được mặc định có kiểu int`
  - `void f(a, b) double a, b; { } // a, b có kiểu double`
  - Khi gọi hàm các đối số không được ép kiểu.
  - Phong cách cổ thường gặp trong các mã nguồn thời đầu.
  - *(Có thể không còn được hỗ trợ trong tương lai).*



# Hàm có danh sách tham số thay đổi

- Phần thay đổi được mô tả bằng dấu ... ở cuối danh sách tham số:
  - Ví dụ: `extern int printf (const char *format, ...);`

*(Nội dung tham khảo thêm)*

# Các liên kết

- Trình biên dịch xác định 1 định danh được khai báo nhiều lần trong 1 chương trình là tên của 1 thực thể hay các thực thể khác nhau dựa trên cơ chế liên kết.
- Các mô tả liên kết cần thiết để đáp ứng các yêu cầu phát triển phần mềm, đặc biệt là các chương trình bao gồm nhiều đơn vị biên dịch.
- Có 3 hình thức liên kết:
  - **Ngoại/external:** 1 định danh được khai báo với liên kết ngoại ở bất kỳ nơi nào trong chương trình đều là tên của 1 thực thể.
  - **Nội/internal:** 1 định danh được khai báo với liên kết nội ở bất kỳ nơi nào trong 1 đơn vị biên dịch đều là tên của 1 thực thể.
  - **Không/none:** 1 định danh được khai báo không liên kết luôn là tên của 1 thực thể duy nhất.

# Liên kết đối với hàm

- Hàm được mặc định (/nếu không có mô tả cụ thể) có liên kết ngoại
- Hàm được khai báo với phân lớp **static** có liên kết nội
  - *(Tham khảo thêm)*
- Hàm **inline** không có liên kết ngoại
  - *(Tham khảo thêm)*

# Khai báo hàm và nguyên mẫu hàm

*Trong khai báo hàm mô tả kiểu trả về là thành phần bắt buộc, mô tả tham số là thành phần không bắt buộc.*

- Khai báo hàm có mô tả kiểu của các tham số hoặc **void** ở vị trí mô tả tham số được gọi là nguyên mẫu hàm.
  - `int f(int, int);` // Nguyên mẫu hàm
  - `int f(void);` // Nguyên mẫu hàm
  - `int f();` // Là khai báo nhưng không phải nguyên mẫu
    - Có thể được sử dụng để gọi hàm có tham số
  - Nguyên mẫu hàm được coi là phong cách C hiện đại và được sử dụng phổ biến hiện nay.
- Định nghĩa hàm đồng thời chứa khai báo hàm.
  - Đoạn mã thu được bằng cách thay thân hàm trong định nghĩa hàm bằng câu lệnh `null` là khai báo hàm, là nguyên mẫu hàm nếu không để trống danh sách kiểu tham số.

## Ví dụ 6.2. Khai báo và nguyên mẫu hàm

// Định nghĩa hàm sum:

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

// Có thể sử dụng các khai báo sau để gọi hàm sum:

```
int sum(int a, int b); // Nguyên mẫu  
int sum(int, int); // Nguyên mẫu  
int sum(); // Không phải nguyên mẫu
```

// Định nghĩa hàm say\_hello()

```
void say_hello() {  
    printf("Hello world!\n");  
}
```

// Có thể sử dụng các khai báo sau để gọi hàm say\_hello

```
void say_hello(void); // Nguyên mẫu  
void say_hello(); // không phải nguyên mẫu
```

# Ví dụ 6.3a. Định nghĩa và khai báo hàm

*Chương trình đổi độ dài được đo bằng inch sang cm.*

```
1  #include <stdio.h>
2  #define INCH1 2.54
3  double inch_to_cm(double x) {
4      return x * INCH1;
5  }
6  int main() { Câu lệnh return trả về giá trị cho
7      double x;
8      printf("Nhập độ dài (inch): ");
9      scanf("%lf", &x); Biểu thức gọi hàm với đối số x
10     printf("%.2f in = %.2f cm\n",
11             x, inch_to_cm(x));
12     return 0;
13 }
```

Nhập độ dài (inch): 5  
5.00 in = 12.70 cm

**Hàm `inch_to_cm` thuộc kiểu hàm trả về (giá trị kiểu) `double`**

## Ví dụ 6.3b. Định nghĩa và khai báo hàm<sup>(2)</sup>

*Có thể đặt định nghĩa hàm trong 1 đơn vị biên dịch khác.*

### **conv.c**

```
1  #include <stdio.h>
2  double inch_to_cm(double x);
3  int main() {
4      double x;
5      printf("Nhập độ dài (inch): ");
6      scanf("%lf", &x);
7      printf("%.2f in = %.2f cm\n",
8              x,      inch_to_cm(x));
9      return 0;
10 }
```

Phát sinh lỗi ở pha ghép  
nối nếu không liệt kê lib.c

**gcc -o p conv.c lib.c**

### **lib.c**

```
1  #define INCH1 2.54
2  double inch_to_cm(double x) {
3      return x * INCH1;
4  }
```

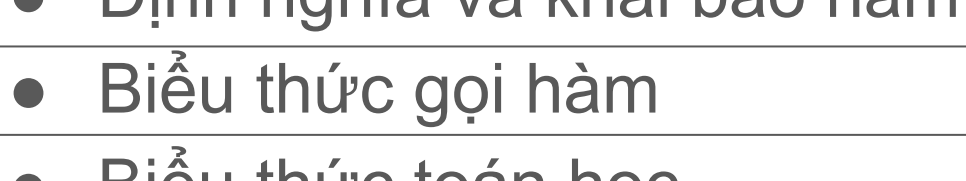
```
Nhập độ dài (inch):
10
10.00 in = 25.40 cm
```

# Tính tương thích của các khai báo

- Trong 1 chương trình C có thể có nhiều khai báo của 1 hàm, các khai báo của 1 hàm không cần phải giống nhau tuyệt đối, nhưng phải tương thích.
  - Khái niệm tương thích khai báo hàm phụ thuộc vào khái niệm tương thích kiểu.
- Trước tiên chúng ta xét 1 số trường hợp thường gặp:
  - Các khai báo hàm có kiểu trả về và kiểu của các tham số giống nhau là các khai báo tương thích.
    - Định danh của các tham số có thể khác nhau nhưng phải cùng kiểu, tên tham số trong khai báo có thể được bỏ qua
    - `double inch_to_cm(double x);` // Tương thích với
    - `double inch_to_cm (double);` //
  - ... Hoặc để rỗng danh sách tham số
    - `double inch_to_cm();` // Phong cách cổ
  - Các khai báo hàm khác kiểu trả về không tương thích.



# Nội dung

- Khái niệm hàm
  - Định nghĩa và khai báo hàm
  - Biểu thức gọi hàm
  - Biểu thức toán học
- 

# Biểu thức gọi hàm

- Định dạng:
  - tên-hàm(danh-sách-đối-số<sub>opt</sub>)
  - biểu-thức-hàm(danh-sách-đối-số<sub>opt</sub>)
- Điều kiện:
  - biểu-thức-hàm có giá trị là con trỏ tới 1 hàm hợp lệ.
    - *(chi tiết về con trỏ hàm được cung cấp sau)*
    - Tên hàm như trong khai báo hàm là biểu thức hàm. Biểu thức hàm chỉ bao gồm tên hàm cho kết quả là con trỏ tới hàm tương ứng.
  - nếu biểu-thức-hàm có kiểu bao gồm nguyên mẫu hàm thì số lượng đối số phải == số lượng tham số,
  - đồng thời các đối số phải thỏa mãn các điều kiện như trong phép gán cho đối tượng thuộc kiểu của tham số.

# Ý nghĩa của biểu thức gọi hàm

- Biểu thức hàm và các đối số được thực hiện và giá trị của đối số được gán cho tham số tương ứng / *Các đối số được truyền **theo giá trị**.* Sau đó thân hàm được thực hiện.
- Thứ tự thực hiện biểu-thức-hàm và các đối số là không xác định, tuy nhiên có 1 điểm tuần tự trước khi thân hàm được thực hiện.
- Nếu hàm được gọi trả về giá trị thì biểu thức gọi hàm có giá trị = giá trị biểu thức trong lệnh **return** trong hàm được gọi và có kiểu là kiểu trả về của hàm. Nếu ngược lại, hàm có kiểu trả về là **void** và biểu thức gọi hàm cũng có kiểu **void**.
- *(Nếu hàm được gọi không tương thích với kiểu được trả tới bởi biểu-thức-hàm thì hành vi là bất định.)*

# Các quy tắc ép kiểu trong gọi hàm

- Nếu *biểu-thức-hàm* có kiểu gắn với nguyên mẫu hàm thì:
  - Các đối số được ép kiểu của tham số tương ứng như trong phép gán. Nếu nguyên mẫu hàm có số lượng tham số thay đổi (có chứa ... như hàm **printf**) thì *quy tắc nâng kiểu đối số* được áp dụng cho các đối số thuộc phần mở rộng (...).
- Nếu *biểu-thức-hàm* có kiểu không gắn với nguyên mẫu hàm:
  - (Ví dụ: Khai báo với danh sách tham số rỗng.)
  - Quy tắc nâng kiểu đối số được áp dụng cho các đối số.
  - Nếu số lượng đối số  $\neq$  số lượng tham số, hoặc kiểu của đối số sau khi được nâng kiểu không tương thích với kiểu của tham số thì hành vi là bất định.
  - => Nên sử dụng nguyên mẫu hàm

[Quy tắc nâng kiểu đối số = Quy tắc nâng kiểu số nguyên + Ép kiểu **double** cho đối số có kiểu **float**.]

# Lệnh return

- Định dạng:
  - **return** ;
  - **return** biểu-thức;
- Điều kiện:
  - **return** ; chỉ được sử dụng trong hàm với kiểu trả về là **void**
  - **return** biểu-thức; - trong hàm trả về giá trị
- Ý nghĩa:
  - Lệnh **return** kết thúc thực hiện hàm đang trực tiếp chứa nó và chuyển điều khiển về vị trí gọi hàm;
  - **return** biểu-thức; thiết lập giá trị biểu-thức trong lệnh **return** là giá trị biểu thức gọi hàm.
  - Nếu biểu-thức có kiểu khác với kiểu trả về của hàm thì giá trị biểu-thức được ép kiểu như trong phép gán cho đối tượng có kiểu là kiểu trả về của hàm.

# Gọi hàm đệ quy

*Biểu thức gọi chính hàm trực tiếp chứa nó trực tiếp hoặc gián tiếp được gọi là gọi hàm đệ quy.*

- Được sử dụng để giải quyết các (lớp) bài toán tương đối đặc biệt, có tính chất đệ quy. Ví dụ:
  - Giải quyết bài toán theo phương pháp chia để trị.
  - Giải thuật sắp xếp nhanh (quick sort)
  - v.v...
- Nên ưu tiên giải quyết bài toán bằng vòng lặp hơn lời giải đệ quy nếu có thể, do các đặc điểm triển khai sử dụng vòng lặp có thể hiệu quả hơn đệ quy.
- Khi cài đặt giải thuật đệ quy cần lưu ý các giới hạn trong triển khai NNLT: Giới hạn bộ nhớ, độ sâu đệ quy, v.v..

# Ví dụ 6.4a. Gọi hàm đệ quy trực tiếp

*Chương trình này ngừng hoạt động khi ngăn xếp gọi hàm bị tràn (lỗi Segmentation fault).*

```
1  #include <stdio.h>
2  void recursion(long n) {
3      printf("n = %ld\n", n);
4      recursion(n+1);
5  }
6  int main() {
7      recursion(1);
8      return 0;
9  }
```

*Phụ thuộc vào môi trường thực thi.*

```
...
n = 261830
n = 261831
n = 261832
Segmentation fault (core dumped)
```

## Ví dụ 6.4b. đệ quy trong chuỗi gọi hàm

*Hàm a và hàm b được gọi đệ quy trong chu trình gọi hàm.*

```
1  #include <stdio.h>
2  void a(long a);
3  void b(long b);
4  void a(long v) {
5      printf("v (a) = %ld\n", v);
6      b(v + 1);
7  }
8  void b(long v) {
9      printf("v (b) = %ld\n", v);
10     a(v + 1);
11 }
12 int main() {
13     a(1);
14     return 0;
15 }
```

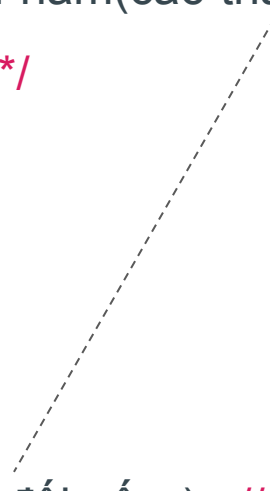
```
v (a) = 261779
v (b) = 261780
v (a) = 261781
Segmentation fault (core dumped)
```



# Tương tác với hàm

*Cấu trúc phổ biến của 1 đơn vị biên dịch*

```
...  
các đối tượng trong phạm vi tệp  
các hàm trong phạm vi tệp  
kiểu-trả-về tên-hàm(các-tham-sốopt) {  
    /* Thân hàm */  
}  
...  
int main() {  
    ...  
    tên-hàm(các-đối-sốopt); // gọi hàm  
    ...  
}
```



- Các tham số của hàm được khởi tạo với các giá trị được cung cấp qua các đối số trong biểu thức gọi hàm.
- Các xử lý trong thân hàm:
  - Có thể sử dụng các thành phần và làm thay đổi giá trị của các đối tượng trong phạm vi tệp.
  - Có thể thay đổi giá trị của tham số. Tuy nhiên thay đổi giá trị tham số không ảnh hưởng đến các đối tượng ngoài hàm.
  - Nếu cần thay đổi giá trị của bất kỳ đối tượng nào thì có thể truyền cho hàm con trỏ tới đối tượng đó (sẽ học sau).

# Ví dụ 6.5a. Truyền tham số cho hàm

*x là biến địa phương trong phạm vi hàm inc10*

```
1  #include <stdio.h>
2  void inc10(int x) {
3      x += 10;
4      printf("(Trong inc10) x = %d\n", x);
5  }
6  int main() {
7      int x = 100;
8      inc10(x);
9      printf("(sau khi gọi inc10) x = %d\n", x);
10     return 0;
11 }
```

*Truyền giá trị của x (100) cho hàm inc10.*

(Trong inc10) x = 110
(Trong main) x = 100

*Giá trị của biến x trong hàm **main** không thay đổi trong trường hợp này.*

## Ví dụ 6.5b. Truyền tham số cho hàm

*x vẫn là biến địa phương trong hàm inc10,  
tuy nhiên là con trỏ trong trường hợp này*

```
1  #include <stdio.h>
2  void inc10(int *x) {
3      *x += 10;
4      printf("(Trong inc10) *x = %d\n", *x);
5  }
6  int main() {
7      int x = 100;
8      inc10(&x);
9      printf("(Trong main) x = %d\n", x);
10 }
```

*Con trỏ tới x được truyền cho hàm inc10*

(Trong inc10) *x = 110 (Trong main) x = 110
--

*Giá trị của biến x trong hàm **main** đã thay đổi trong trường hợp này.*

## Ví dụ 6.5c. Đối tượng có phạm vi tệp

*x có phạm vi tệp, là biến toàn cục*

```
1  #include <stdio.h>
2  int x;
3  void inc10() {
4      x += 10;
5  }
6  int main() {
7      x = 100;
8      inc10();
9      printf("(sau inc10()) x = %d\n", x);
10 }
```

*Sử dụng chung 1 biến x toàn cục*

(Trong main) x = 110

*Giá trị của biến x (có phạm vi tệp) đã thay đổi trong trường hợp này.*

# Thứ tự ưu tiên và chiều thực hiện chuỗi toán tử

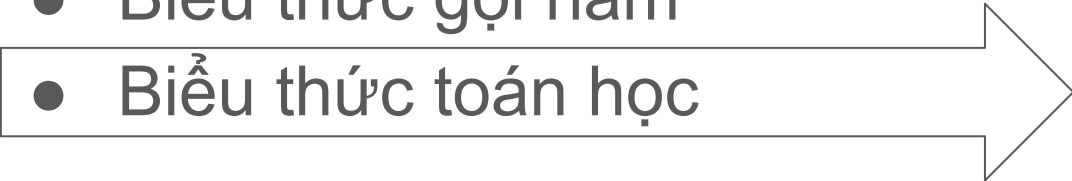
Ưu tiên cao  
(thực hiện trước)



Ưu tiên thấp  
(thực hiện sau)

Tên toán tử	Ký hiệu	Ví dụ	Chuỗi
Gọi hàm Tăng 1 (hậu tố) Giảm 1 (hậu tố)	() ++ --	f(3, 5) x++ x--	⇒
Tăng, giảm 1 (tiền tố) Dấu (tiền tố) Đảo dây bit Phủ định Dung lượng	++, -- +, - ~ ! sizeof	++x, --x +x, -x ~x !x sizeof(int)	⇐
Ép kiểu	()	(double)n	⇐
Nhân, Chia, Phần dư	*, /, %	x * y, x / y, x % y	⇒
Cộng, trừ	+, -	x + y, x - y	⇒
Dịch trái, dịch phải	<<, >>	x << y, x >> y	⇒
So sánh thứ tự	<, >, <=, >=	x < y	⇒
So sánh bằng	==, !=	x == y	⇒
AND theo bit	&	x & y	⇒
XOR theo bit	^	x ^ y	⇒
OR theo bit		x   y	⇒
AND lô-gic	&&	x && y	⇒
OR lô-gic		x    y	⇒
Lựa chọn	?:	x ? y: z	⇐
Gán đơn giản Gán kết hợp	= *=, /=, %=, +=, -=, <=<=, >>=, &=, ^=,  =	x = y x *= y, x /= y, ...	⇐

# Nội dung

- Khái niệm hàm
  - Định nghĩa và khai báo hàm
  - Biểu thức gọi hàm
  - Biểu thức toán học
- 

# Các hằng toán học <math.h>

```
# define M_E      2.7182818284590452354 /* e */
# define M_PI     3.14159265358979323846 /* pi */
# define M_PI_2   1.57079632679489661923 /* pi/2 */
# define M_SQRT2  1.41421356237309504880 /* sqrt(2) */
...
```

# Các hàm toán học <math.h>

## Các hàm lượng giác, giá trị góc x được đo bằng radian

```
double cos(double x);  
double sin(double x);  
double tan(double x);  
...
```

## Hàm mũ và logarit

```
double exp(double x); //  $e^x$   
double pow(double x, double y); //  $x^y$   
double sqrt(double x); //  $\sqrt{x}$   
double log(double x); //  $\log_e x$   
double log10(double x); //  $\log_{10} x$   
double fabs(double x); //  $|x|$   
...
```



## Ví dụ 6.6. Sử dụng <math.h>

Chương trình tính cạnh huyền của 1 tam giác vuông

**tg.c**

```
1  #include <math.h>
2  #include <stdio.h>
3  int main() {
4      double a, b;
5      printf("Nhập 2 cạnh góc vuông a b: ");
6      scanf("%lf%lf", &a, &b);
7      printf("Độ dài cạnh huyền = %g\n",
8              sqrt(a * a + b * b));
9  }
```

*liên kết với thư viện libm*

**gcc -o p tg.c -lm**

```
Nhập 2 cạnh góc vuông a b: 3 4
Độ dài cạnh huyền = 5
```

