

# Ngôn ngữ lập trình C

## Bài 5. Hàm

*Soạn bởi: TS. Nguyễn Bá Ngọc*

# Các nội dung đã học về hàm

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

- Hàm main như chúng ta đã viết trong các chương trình là một hàm hoàn chỉnh. Hàm main còn là 1 hàm đặc biệt:
  - Được quy ước là điểm bắt đầu thực hiện chương trình.
  - Không được khai báo inline hoặc static.
  - Mỗi chương trình có đúng 1 hàm main.
- Chúng ta cũng đã sử dụng các hàm trong thư viện chuẩn:
  - **printf** - Hàm xuất dữ liệu theo định dạng.
  - **scanf** - Hàm nhập dữ liệu theo định dạng.

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

# Ví dụ 5-1a. Vấn đề trùng lặp mã nguồn

Chúng ta cùng phân tích 1 vấn đề tính USCLN

```
9  #include <stdio.h>
10 int main() {
11     int a, b;
12     printf("Nhập a b: ");
13     scanf("%d%d", &a, &b);
14     while (b != 0) {
15         int tmp = a % b;
16         a = b;
17         b = tmp;
18     }
19     printf("USCLN = %d\n", a);
20     return 0;
21 }
```

```
bangoc:$gcc -o prog vd5-1a.c
bangoc:$./prog
Nhập a b: 18 30
USCLN = 6
bangoc:$
```

**Thử:** điều chỉnh chương trình để nhập 3 số a, b, c rồi in ra các USCLN của a và b, b và c, và a và c?

*Sao lưu các giá trị a, b, c rồi lặp lại mã nguồn tính USCLN nhiều lần cho các cặp giá trị không phải ý tưởng hay, sử dụng hàm sẽ hiệu quả hơn.*

## Ví dụ 5-1b. Hàm tính USCLN

*Có thể tham khảo lời giải cho vấn đề trong Ví dụ 5-1a.*

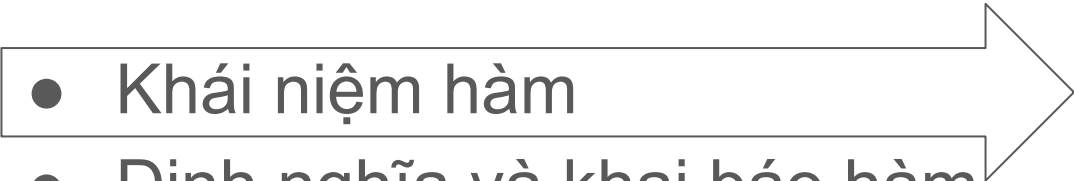
```
5 #include <stdio.h>
6 int uscln(int a, int b) {
7     while (b != 0) {
8         int tmp = a % b;
9         a = b;
10        b = tmp;
11    }
12    return a;
13 }
14 int main() {
15     int a, b, c;
16     printf("Nhập a b c: ");
17     scanf("%d%d%d", &a, &b, &c);
18     printf("USCLN(%d, %d) = %d\n", a, b, uscln(a, b));
19     printf("USCLN(%d, %d) = %d\n", a, c, uscln(a, c));
20     printf("USCLN(%d, %d) = %d\n", b, c, uscln(b, c));
21     return 0;
22 }
```

bangoc:\$gcc -o prog vd5-1b.c  
bangoc:\$./prog  
Nhập a b c: 18 30 27  
USCLN(18, 30) = 6  
USCLN(18, 27) = 9  
USCLN(30, 27) = 3  
bangoc:\$

# 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 sử dụng nhiều lần ở nhiều nơi mà không cần lặp lại mã nguồn.
- Hàm ẩn các chi tiết triển khai, người lập trình có thể sử dụng hàm như thao tác bậc cao, cung cấp các dữ liệu đầu vào và nhận các kết quả đầu ra.
- Với hàm 1 vấn đề tính toán lớn có thể được chia nhỏ và xử lý từng phần bằng các hàm bởi nhiều người.

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

*Khái niệm hàm trong NNLT có một số đặc điểm chung nhưng trong tổng thể thì khác với khái niệm hàm trong toán học*

# Kiểu hàm

*Kiểu hàm mô tả hàm với kiểu trả về, số lượng và kiểu của các tham số*

- Kiểu hàm được coi là kiểu suy diễn của kiểu dữ liệu mà hàm trả về.
  - `int f(int a, int b);` `f` là hàm trả về `int` ...
- Tên kiểu hàm được viết giống như khai báo hàm nhưng lược bỏ định danh.
  - Thường được sử dụng để tạo 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

# Các yêu cầu định nghĩa & khai báo hàm

*NNLT C tách rời 2 khái niệm định nghĩa hàm và khai báo hàm. Trong đó định nghĩa hàm bao gồm cả khai báo hàm và thân hàm. Khai báo hàm không bao gồm 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 cần có khai báo hàm.
  - Hàm có thể được định nghĩa trong cùng 1 đơn vị biên dịch với biểu thức gọi hàm hoặc trong 1 đơn vị biên dịch khác
  - Trường hợp hàm được sử dụng nhưng không có định nghĩa / trình biên dịch không tìm thấy định nghĩa sẽ làm phát sinh lỗi ở pha lỗi ghép nối.
- Hàm được định nghĩa và khai báo trong phạm vi tệp.

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

# Mã nguồn định nghĩa & khai báo hàm

- 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.
  - Đảm bảo tính nhất quán trong nhiều đơn vị biên dịch.
- Mỗi hàm thường được định nghĩa 1 lần và ở dạng mã nguồn 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 hàm inline và 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: Riêng trong phạm vi đơn vị biên dịch.
  - Phải xác định được 1 định nghĩa duy nhất cho mỗi biểu thức gọi hàm.

# Cấu trúc định nghĩa hàm

*Thông thường hàm được định nghĩa theo cấu trúc:*

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

- *kiểu-trả-về*: Hàm có thể trả về 1 giá trị kiểu số hoặc kiểu khác do người lập trình tự định nghĩa (*sẽ học sau*):
  - Nếu hàm không trả về giá trị thì kiểu trả về được để là **void**.
  - Nếu không có mô tả 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) sẽ có phạm vi khối và có hiệu lực trong thân hàm.
- **câu-lệnh-gộp**: Thân hàm, được thực hiện sau khi hàm được gọi và giá trị của các tham số đã được thiết lập.

# Hàm không có tham số

- Trong định nghĩa hàm chúng ta có thể để trống danh sách tham số hoặc sử dụng từ khóa void (có cùng ý nghĩa).
- Trong khai báo hàm từ khóa void được sử dụng để mô tả hàm không có tham số.
- Để trống danh sách tham số () có thể có nhiều ý nghĩa:
  - Trong định nghĩa: Danh sách trống = Không có tham số
  - Trong khai báo: Danh sách 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();` // - hàm foo2 có thể không có tham số hoặc có tham số

# Hàm có danh sách tham số cố định

- Hiện nay hình thức khai báo từng tham số / danh sách kiểu được sử dụng phổ biến:
  - `void f(int a, int b, int c) { /* ... */ } // OK`
  - `void f(int a, b, c) { /* ... */ } // NOK-Cần khai báo từng tham số`
  - Khi gọi hàm các đối số được ép kiểu của tham số.
  - Được coi là phong cách C 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 (phong cách cổ):
  - Ví dụ: `void f(a, b, c) { /* ... */ } // Được mặc định có kiểu int`
  - `void f(a, b) double a, b; { /* .. */ } // a, b có kiểu double`
  - Không ép kiểu các đối số trong biểu thức gọi hàm.
  - *(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 trong mỗi khai báo là tên của 1 thực thể duy nhất.



# Liên kết đối với hàm

- Hàm được mặc định (/ nếu không có mô tả liên quan) 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 & nguyên mẫu hàm

*Khai báo hàm mô tả kiểu trả về của hàm và có thể không mô tả tham số.*

- Khai báo hàm có mô tả kiểu của các tham số được gọi là nguyên mẫu hàm.
  - `int f(int, int);` // Nguyên mẫu hàm
  - `int f(void);` // -
  - `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. Mô tả thu được bằng cách thay thân hàm trong định nghĩa hàm bằng câu lệnh `null` là khai báo hàm/có thể là nguyên mẫu.

## Ví dụ 5-2. Khai báo & 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 5-3a. Định nghĩa hàm

Chương trình này sử dụng hàm để quy đổi độ dài được đo bằng inch sang cm.

```
6 #include <stdio.h>
```

```
7
```

```
8 #define INCH1 2.54
```

Nguyên mẫu hàm

```
9  
10 double inch_to_cm(double x) {  
11     return x * INCH1;  
12 }  
13
```

```
bangoc:$gcc -o prog vd6-1.c
```

```
bangoc:$./prog
```

```
Nhập độ dài (inch): 5
```

```
5.00 in = 12.70 cm
```

```
bangoc:$
```

Định nghĩa hàm `inch_to_cm`

- Trả về giá trị có kiểu `double`

- Nhận 1 tham số có kiểu `double`

Đóng gói và ẩn cách chuyển đổi inch => cm

```
14 int main() {
```

```
15     double x;
```

```
16     printf("Nhập độ dài (inch): ");
```

```
17     scanf("%lf", &x);
```

```
18     printf("%.2f in = %.2f cm\n", x, inch_to_cm(x));
```

```
19     return 0;
```

```
20 }
```

Câu lệnh `return` trả về giá trị cho biểu thức gọi hàm

Biểu thức gọi hàm với đối số `x`

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

# Ví dụ 5-3b. Định nghĩa và khai báo hàm

Chúng ta tái cấu trúc ví dụ 5.2, chuyển định nghĩa hàm `inch_to_cm` sang 1 đơn vị biên dịch khác.

```
vd6-2a.c
6 #include <stdio.h>
7
8 double inch_to_cm(double x);
9
10 int main() {
11     double x;
12     printf("Nhập độ dài (inch): ");
13     scanf("%lf", &x);
14     printf("%.2f in = %.2f cm\n", x, inch_to_cm(x));
15     return 0;
16 }
```

Khai báo và cũng là nguyên mẫu của hàm `inch_to_cm`

Nguyên mẫu hàm = Khai báo hàm với mô tả kiểu của tham số.

Phát sinh lỗi ở pha ghép nối nếu không liệt kê `vd6-2b.c`

```
bangoc:$gcc -o prog vd6-2a.c vd6-2b.c
bangoc:$./prog
Nhập độ dài (inch): 3
3.00 in = 7.62 cm
bangoc:$
```

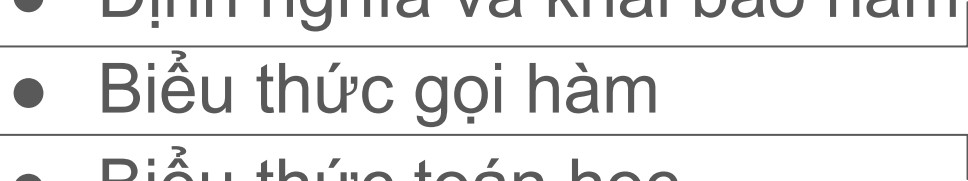
```
vd6-2b.c
1 #define INCH1 2.54
2
3 double inch_to_cm(double x) {
4     return x * INCH1;
5 }
```

Hàm được mặc định có liên kết ngoại (extern) và có thể được sử dụng (gắn kết với khai báo) trong đơn vị biên dịch khác.

# 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 (*chi tiết được cung cấp sau*).
- 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 kiểu đối tượng thì biểu thức có giá trị = giá trị biểu thức trong lệnh **return** trong hàm và có kiểu là kiểu trả về của hàm. Nếu ngược lại thì hàm trả về kiểu **void** và biểu thức có kiểu **void**.
- *(Nếu hàm được gọi không tương thích với kiểu được trả về 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 sau khi hiệu chỉnh thì hành vi là bất định.
  - Nên sử dụng nguyên mẫu

[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 (kiểu khác void)
- 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

*Trong C một hàm có thể gọi chính nó trực tiếp hoặc gián tiếp, các biểu thức gọi hàm như vậy 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 do các hạn chế của cơ chế đệ quy.
- Trong cài đặt giải thuật đệ quy cần lưu ý các giới hạn triển khai NNLT: Giới hạn bộ nhớ, độ sâu đệ quy, v.v..

# Ví dụ 5-4a. Minh họa gọi đệ quy

*Chương trình này gọi đệ quy hàm recursion và in ra giá trị biến n - cho biết chiều sâu đệ quy. Chương trình ngừng hoạt động khi ngăn xếp gọi hàm bị tràn (lỗi Segmentation fault).*

```
6  #include <stdio.h>
7
8  void recursion(long n) {
9      printf("n = %ld\n", n);
10     recursion(n+1);
11 }
12
13 int main() {
14     recursion(1);
15     return 0;
16 }
```

```
n = 261727
n = 261728
n = 261729
n = 261730
n = 261731
Segmentation fault (core dumped)
bangoc:$
```

*Kết quả xuất ra màn hình phụ thuộc vào môi trường thực thi.*

# Ví dụ 5-4b. Minh họa đệ quy trong chuỗi gọi hàm

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

```
vd6-5b.c
6  #include <stdio.h>
7
8  void a(long a);
9  void b(long b);
10
11 void a(long v) {
12     printf("v (a) = %ld\n", v);
13     b(v + 1);
14 }
15
16 void b(long v) {
17     printf("v (b) = %ld\n", v);
18     a(v + 1);
19 }
20
21 int main() {
22     a(1);
23     return 0;
24 }
```

```
v (a) = 261875
v (b) = 261876
v (a) = 261877
v (b) = 261878
Segmentation fault (core dumped)
bangoc:$
```

*Kết quả xuất ra màn hình phụ thuộc vào môi trường thực thi.*

*Chương trình này ngừng hoạt động do phát sinh lỗi tràn ngăn xếp bộ nhớ gọi hàm.*

# Tương tác với hàm

## Cấu trúc thông thường 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:
  - Làm thay đổi giá trị của tham số. Tuy nhiên thay đổi giá trị tham số không ảnh hưởng đến các đối tượng ngoài hàm.
  - Có thể làm thay đổi giá trị của các đối tượng và sử dụng các thành phần trong phạm vi tệp trong cùng đơn vị biên dịch hoặc trong đơn vị biên dịch khác.

*Nếu cần thay đổi giá trị của đối tượng (bất kỳ / bên ngoài hàm) thì có thể truyền cho hàm con trỏ tới đối tượng đó (sẽ học sau).*

# Ví dụ 5.5a. Truyền tham số cho hàm

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

```
6 #include <stdio.h>
7
8 void inc10(int x) {
9     x += 10;
10    printf("(Trong inc10) x = %d\n", x);
11 }
12
13 int main() {
14     int x = 100;
15     inc10(x);
16     printf("(sau khi gọi inc10) x = %d\n", x);
17     return 0;
18 }
```

*Giá trị của x được truyền cho hàm inc10.*

```
bangoc:$gcc -o pa vd6-3a.c
bangoc:$./pa
(Trong inc10) x = 110
(sau khi gọi inc10) x = 100
bangoc:$
```

*Thực hiện hàm inc10 không làm thay đổi giá trị biến x trong hàm main trong trường hợp này.*



# Ví dụ 5.5b. Truyền tham số cho hàm

*Tham số x vẫn là biến địa phương của hàm inc10, tuy nhiên x là con trỏ và trỏ tới x trong main*

```
6 #include <stdio.h>
7
8 void inc10(int *x) {
9     *x += 10;
10    printf("(Trong inc10) *x = %d\n", *x);
11 }
12
13 int main() {
14     int x = 100;
15     inc10(&x);
16     printf("(sau khi gọi inc10) x = %d\n", x);
17     return 0;
18 }
```

```
bangoc:$gcc -o prog vd6-3b.c
bangoc:$./prog
(Trong inc10) *x = 110
(sau khi gọi inc10) x = 110
bangoc:$
```

*Giá trị của biểu thức &x và là địa chỉ của biến x được truyền cho hàm inc10.*

*Thực hiện hàm inc10 làm thay đổi giá trị của biến x trong trường hợp này.*

## Ví dụ 5.5c. Đối tượng trong phạm vi tệp

*x được khai báo trong phạm vi tệp và có thể được sử dụng ở nhiều nơi khác nhau*

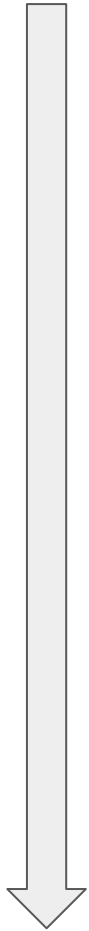
```
5 #include <stdio.h>
6 int x;
7 void inc10() {
8     x += 10;
9 }
10 int main() {
11     x = 100;
12     inc10();
13     printf("(sau inc10()) x = %d\n", x);
14     return 0;
15 }
```

```
bangoc:$gcc -o prog vd5-5c.c
bangoc:$./prog
(sau inc10()) x = 110
bangoc:$
```

*Thực hiện hàm inc10 làm thay đổi giá trị của biến x 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	Thứ tự
<b>Gọi hàm</b> Tăng 1, giảm 1 (hậu tố)	<b>()</b> <b>f(3, 5)</b> ++, --    (x++, x--)	Trái -> Phải
Tăng, giảm 1 (tiền tố) Dấu (tiền tố) Phủ định lô-gic sizeof	++, --    (++x, --x) +, -      (+x, -x) !        (!e) sizeof    (sizeof(int))	Phải -> Trái
<b>Ép kiểu</b>	<b>()</b> <b>(double)5</b>	Phải -> Trái
Nhân, Chia Phần dư	*, /      (x * y, x / y) %        (x % y)	Trái -> Phải
Cộng Trừ	+        (x + y) -        (x - y)	Trái -> Phải
Dịch sang trái Dịch sang phải	<<      (x << y) >>      (x >> y)	Trái -> Phải
So sánh thứ tự	<, >, <=, >=	Trái -> Phải
So sánh bằng	==, !=	Trái -> Phải
AND theo bit	&        (x & y)	Trái -> Phải
XOR theo bit	^        (x ^ y)	Trái -> Phải
OR theo bit	(x   y)	Trái -> Phải
AND lô-gic	&&	Trái -> Phải
OR lô-gic		Trái -> Phải
Lựa chọn	?:	Phải -> Trái (Rẽ nhánh)
Gán đơn giản Gán kết hợp	= *=, /=, %=, +=, -=, <<=, >>=, &=, ^=,  =	Phải -> Trái

# 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 5.6. Sử dụng <math.h>

Chương trình tính cạnh huyền tam giác vuông

```
5 #include <math.h>
6 #include <stdio.h>
7 int main() {
8     double a, b;
9     printf("Nhập 2 cạnh góc vuông a b: ");
10    scanf("%lf%lf", &a, &b);
11    printf("Độ dài cạnh huyền = %g\n",
12           sqrt(a * a + b * b));
13    return 0;
14 }
```

```
bangoc:$gcc -o prog vd5-6.c -lm
bangoc:$./prog
Nhập 2 cạnh góc vuông a b: 3 4
Độ dài cạnh huyền = 5
bangoc:$
```

Để biên dịch chương trình có sử dụng hàm toán học trong <math.h> với GCC chúng ta cần liên kết với thư viện libm bằng cách bổ xung -lm vào câu lệnh biên dịch.

