

Ngôn ngữ lập trình C

Bài 4. Biểu thức, nhập và xuất theo định dạng

Soạn bởi: TS. Nguyễn Bá Ngọc

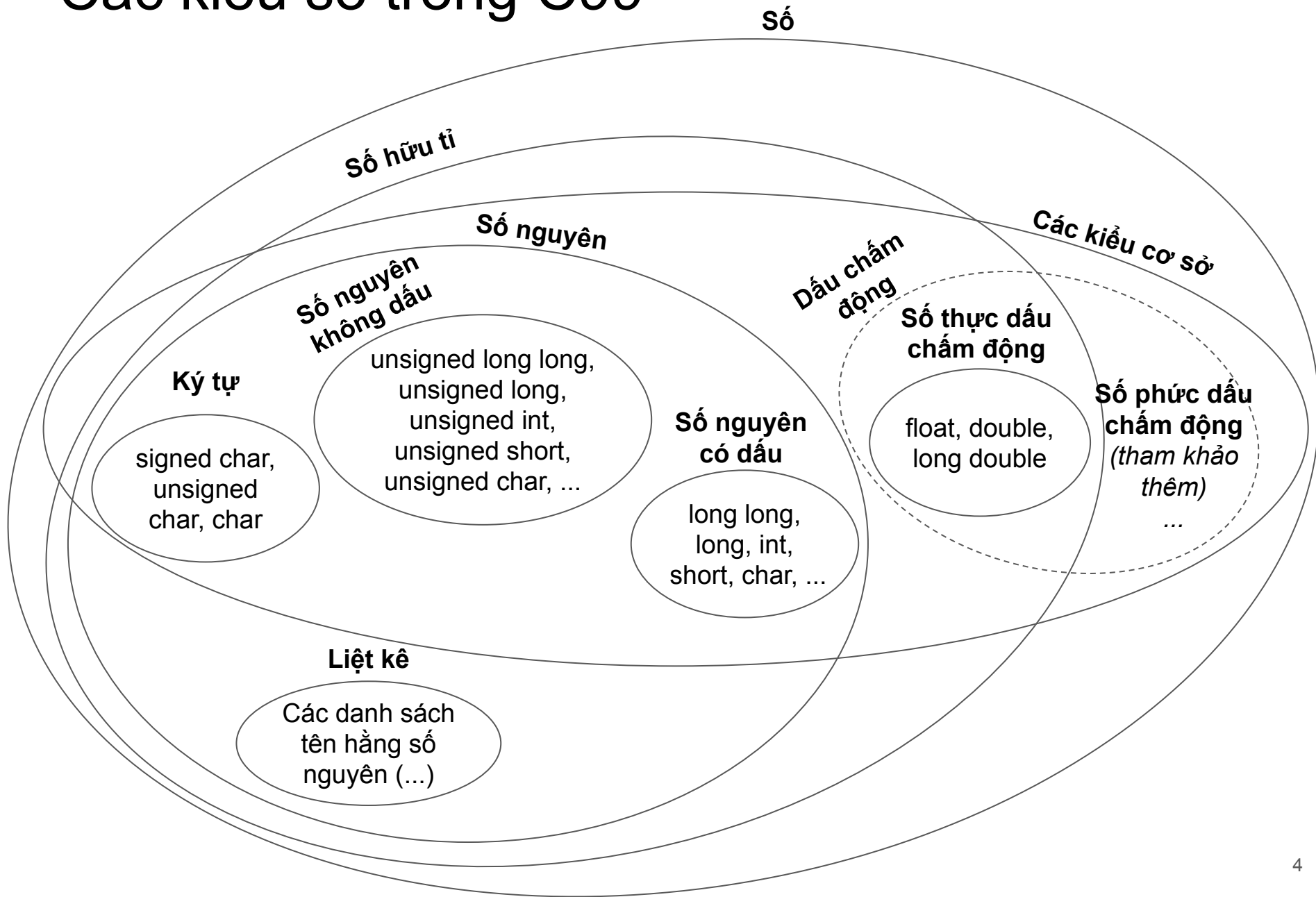
Nội dung

- Biểu thức
- Nhập và xuất theo định dạng

Nội dung

- 
- Biểu thức
 - Nhập và xuất theo định dạng

Các kiểu số trong C99



Tổng quan về biểu thức trong C

- Trong C định danh, hằng giá trị, và biểu thức thu được bằng cách đặt một biểu thức trong cặp dấu ngoặc đơn () là những biểu thức cơ bản.
 - Biểu thức dạng (E) có kiểu và giá trị giống như biểu thức E.
- Các biểu thức có thể được kết hợp bằng các toán tử theo quy tắc để tạo thành các biểu thức lớn hơn.
 - Trong chương này chúng ta sẽ học một số toán tử thông dụng cho kiểu số. Các toán tử khác sẽ được khám phá dần theo tiến trình học.
- Khác với khái niệm toán tử trong toán học, toán tử trong C có thể có hiệu ứng phụ (ví dụ làm thay đổi giá trị của biến).
 - Chúng ta chỉ nên thay đổi giá trị của 1 biến tối đa 1 lần trong 1 biểu thức.
 - *(Tham khảo thêm về đơn vị tuần tự/sequence point).*

Biểu thức và câu lệnh

- Biểu thức có thể là thành phần của câu lệnh, trong trường hợp đó biểu thức được tính khi câu lệnh được thực hiện.
- Câu lệnh có thể chỉ bao gồm biểu thức và dấu ; - chúng ta gọi các câu lệnh như vậy là câu lệnh tính biểu thức.
 - Câu lệnh cũng có thể chỉ bao gồm dấu ; (được gọi là câu lệnh null - không làm gì cả, nhưng cần thiết để đáp ứng yêu cầu cú pháp trong một số trường hợp).
 - Trong chương này chúng ta chủ yếu làm việc với câu lệnh tính biểu thức.
 - *(Các câu lệnh khác cũng sẽ được khám phá sau theo tiến trình học phần).*

Các toán tử cộng, trừ

Nếu các toán hạng có kiểu số thì quy tắc ép kiểu phổ thông được áp dụng. *(trường hợp 1 toán hạng có kiểu con trỏ sẽ được học sau):*

- Toán tử cộng (+)
 - Cho kết quả là tổng của 2 toán hạng.
 - Ví dụ: $1 + 2.0 \Rightarrow 3.0$ kiểu double
- Toán tử trừ (-)
 - Cho kết quả là hiệu của 2 toán hạng.
 - Ví dụ: $3 - 1 \Rightarrow 2$ kiểu int

Chuỗi các toán tử cộng/trừ được thực hiện từ trái sang phải.

Các toán tử dấu

Toán hạng phải có kiểu số. Quy tắc nâng kiểu số nguyên được áp dụng:

- Toán tử dấu +:
 - Cho kết quả là giá trị của toán hạng (sau khi chuyển kiểu)
 - Ví dụ:
signed char ch = 3;
+ch => 3 kiểu int
- Toán tử dấu - :
 - Cho kết quả là giá trị của toán hạng (đã chuyển kiểu) sau khi đảo dấu
 - Ví dụ:
signed char ch = -5;
-ch => 5 kiểu int.

Các toán tử dấu là các toán tử 1 ngôi dạng tiền tố.

Các toán tử nhân, chia, và phần dư

Các toán hạng phải có kiểu số. Với toán tử % các toán hạng phải có kiểu số nguyên. Quy tắc ép kiểu phổ thông được áp dụng cho các toán hạng:

- Toán tử nhân (*): Kết quả là tích của 2 toán hạng.
 - Ví dụ: $2 * 1.5 \Rightarrow 3.0$ kiểu double
- Toán tử chia (/): Nếu có toán hạng kiểu số thực dấu chấm động thì kết quả là thương của 2 toán hạng, nếu ngược lại (2 số nguyên) thì kết quả là phần nguyên của phép chia.
 - Chia cho 0 là hành vi không xác định.
 - Ví dụ: $3/2 \Rightarrow 1$ kiểu int; $3.0/2 \Rightarrow 1.5$ kiểu double
- Toán tử phần dư (%): Kết quả là phần dư của phép chia.
Ví dụ, $15 \% 6 \Rightarrow 3$ kiểu int.

Chuỗi các toán tử nhân/chia/phần dư được thực hiện theo thứ tự từ trái sang phải.

Phần nguyên và phần dư trong phép chia

Với a, b là các số nguyên trong đó $b \neq 0$ chúng ta có:

$$a = (a / b) * b + a \% b, \text{ bên cạnh đó:}$$

- Trong C89 kết quả phép chia với 2 toán hạng là các số nguyên trong đó có toán hạng là số âm phụ thuộc vào triển khai (có thể được làm tròn lên hoặc làm tròn xuống).
 - $7 / (-3)$ có thể cho kết quả -2 hoặc -3 tùy theo triển khai.
 - *(Phản ánh đúng hiện trạng triển khai của phần cứng.)*
- Trong C99 kết quả phép chia trong tình huống tương tự được quy ước làm tròn về phía giá trị 0 (bằng kết quả chia thông thường, và là hành vi xác định), ví dụ:
 - $7 / (-3) \Rightarrow -2$ kiểu int, và $7 \% (-3) \Rightarrow 1$ kiểu int
 - $(-7) / 3 \Rightarrow -2$ kiểu int, và $(-7) \% 3 \Rightarrow -1$ kiểu int.

Lưu ý chung đối với các toán tử trên dãy bits

Các toán tử trên dãy bits bao gồm: \gg , \ll , $\&$, $|$, \wedge , \sim . Các chi tiết sẽ được cung cấp trong các trang tiếp theo.

Các toán hạng của các toán tử trên dãy bits phải có kiểu số nguyên: Có thể là số nguyên không dấu hoặc số nguyên có dấu. Tuy nhiên *chỉ nên thực hiện các xử lý trên dãy bits với toán hạng có kiểu không dấu. Không nên thực hiện các xử lý trên dãy bits với toán hạng thuộc kiểu có dấu, đặc biệt là các toán hạng có giá trị là các số âm do nhiều hành vi bất định và hành vi phụ thuộc triển khai có thể phát sinh. Bên cạnh đó quy chuẩn C không quy ước phương pháp biểu diễn số nguyên có dấu (tuy hầu hết các trình biên dịch hiện đang sử dụng mã bù-2).*

Các toán tử dịch chuyển dãy bits

Các toán hạng phải có kiểu số nguyên. Quy tắc nâng kiểu số nguyên được áp dụng cho các toán hạng.

- Toán tử dịch chuyển dãy bits sang trái: \ll
- Toán tử dịch chuyển dãy bits sang phải: \gg
- Giả sử sau khi nâng kiểu thì vế trái có kiểu T được biểu diễn bằng n bits:
 - Nếu toán hạng ở vế phải có giá trị $\geq n$ thì hành vi là không xác định.
 - *(Biểu thức được thực hiện tương tự như đã mô tả trong chương 2 với dãy n bits, kết quả có kiểu T).*

Chuỗi các toán tử dịch chuyển dãy bits được thực hiện theo thứ tự từ trái sang phải.

Các toán tử AND, OR, và XOR trên dãy bits

Các toán hạng phải có kiểu số nguyên. Quy tắc chuyển kiểu phổ thông được áp dụng.

- Toán tử và/AND trên dãy bits: &
- Toán tử hoặc/OR trên dãy bits: |
- Toán tử hoặc loại trừ/XOR trên dãy bits: ^
- Giả sử sau khi chuyển kiểu các toán hạng có kiểu T được biểu diễn bằng n bits:
 - *(Biểu thức được thực hiện tương tự như đã mô tả trong chương 2 với dãy n bits, kết quả có kiểu T)*

Chuỗi các toán tử AND/OR/XOR được thực hiện theo thứ tự từ trái sang phải.

Toán tử đảo dãy bits (~)

Toán hạng phải có kiểu số nguyên. Quy tắc nâng kiểu số nguyên được áp dụng.

- Giả sử sau khi áp dụng quy tắc nâng kiểu, toán hạng có kiểu T được biểu diễn bằng n bits:
 - *(Biểu thức được thực hiện tương tự như đã mô tả trong chương 2 với dãy n bits, kết quả có kiểu T).*

Toán tử đảo dãy bits là toán tử 1 ngôi dạng tiền tố.

Các toán tử gán và lvalue

Toán hạng ở bên trái toán tử gán phải là *lvalue* có thể thay đổi. (Động cơ: Cần thay đổi giá trị một vùng nhớ => cần biết địa chỉ vùng nhớ, có thể ghi dữ liệu, tương thích kiểu, v.v.).

- Toán tử gán lưu giá trị vào đối tượng được xác định bởi toán hạng ở vế trái.
- Kết quả của phép gán là giá trị của toán hạng ở vế trái sau khi được gán giá trị, nhưng không phải lvalue.
- Kiểu của biểu thức là kiểu gốc (unqualified type) của toán hạng ở vế trái.
- Tương tự như các toán tử khác, toán tử gán cũng có thể được sử dụng theo chuỗi:
 - Ví dụ: $x = y = (z + 100);$
 - !An toàn với điều kiện x, y, z là các biến không chồng lấn.

Chuỗi các toán tử gán được thực hiện từ phải sang trái.

Các toán tử gán và lvalue₍₂₎

- *lvalue* có thể thay đổi thường dùng nhất chính là biến có kiểu số như chúng ta đã gặp cho tới thời điểm này. Các lvalue khác sẽ được khám phá dần theo tiến trình học.
- Ví dụ lvalue:
 - `int x, y; // x, y, z đều là các lvalue có thể thay đổi`
 - `float z; //`
- Các toán tử gán được chia thành 2 nhóm:
 - Đơn giản: `=`, và
 - Kết hợp: `*=`, `/=`, `%=`, `+=`, `-=`, `<<=`, `>>=`, `&=`, `^=`, `|=`

Các biểu thức gán

- Biểu thức gán đơn giản (với toán tử $=$) thường được sử dụng để lưu giá trị của một biểu thức vào 1 biến.
 - Giá trị toán hạng ở vế phải được chuyển kiểu thành kiểu của biểu thức gán. (*Tham khảo thêm các ràng buộc về kiểu đối với các toán hạng*).
- Các phép gán đơn giản có dạng $e1 = e1 \text{ op } e2$, có thể được viết lại bằng các toán tử gán kết hợp theo định dạng $e1 \text{ op} = e2$, trong đó $\text{op} \in \{*, /, \%, +, -, <<, >>, \&, ^, | \}$
 - Biểu thức gán kết hợp ($e1 \text{ op} = e2$) và biểu thức gán đơn giản tương ứng ($e1 = e1 \text{ op } e2$) về cơ bản là tương đương.
 - Chỉ có 1 khác biệt nhỏ: Trong biểu thức gán kết hợp thì toán hạng $e1$ (ở vế trái) chỉ được tính 1 lần.
 - Các điều kiện về kiểu cho $e1 \text{ op } e2$ cũng được áp dụng cho biểu thức gán kết hợp $e1 \text{ op} = e2$.

Các toán tử tăng và giảm 1

Yêu cầu: Toán hạng là *lvalue* có thể thay đổi.

Có 2 định dạng đối với toán tử tăng 1 (++) và toán tử giảm 1 (--): Hậu tố và tiền tố.

- Định dạng hậu tố:
 - $x++$: Giá trị của x được tăng lên 1, giá trị biểu thức bằng giá trị ban đầu của x (giống như $(tmp = x, x += 1, tmp)$).
 - $x--$: Giá trị của x bị giảm đi 1, giá trị biểu thức bằng giá trị ban đầu của x (giống như $(tmp = x, x -= 1, tmp)$).
- Các toán tử tăng và giảm 1 dạng tiền tố:
 - $++x$: Giá trị của x được tăng lên 1, giá trị biểu thức bằng giá trị của x sau khi tăng (tương đương với $(x += 1)$).
 - $--x$: Giá trị của x được giảm đi 1, giá trị biểu thức bằng giá trị của x sau khi giảm (tương đương với $(x -= 1)$).

Các toán tử tăng/giảm 1 là các toán tử 1 ngôi.

Toán tử sizeof

Toán hạng của sizeof có thể là biểu thức hoặc tên kiểu được đặt trong cặp dấu ngoặc đơn.

- sizeof trả về kích thước tính bằng bytes của toán hạng được xác định dựa trên kiểu của toán hạng.
 - *(không áp dụng sizeof cho các kiểu chưa hoàn hiện).*
- Kết quả của toán tử sizeof có kiểu là kiểu số nguyên không dấu được định nghĩa bởi triển khai cụ thể, được đặt tên là `size_t` và được định nghĩa trong `<stddef.h>`.
- Một số ví dụ:
 - `sizeof(char); // => 1`
 - `sizeof 10; // Phụ thuộc vào triển khai`
 - `sizeof 10.2; // => 8`
 - `sizeof(double); // => 8`

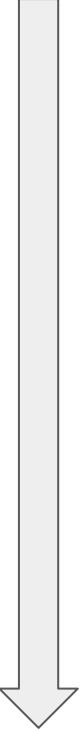
Toán tử sizeof là toán tử 1 ngôi dạng tiền tố.

Quy tắc tính biểu thức phức tạp

- Trong biểu thức phức tạp, các biểu thức thành phần được tạo thành từ các toán tử có độ ưu tiên cao hơn được coi như các toán hạng của các biểu thức thành phần được tạo thành từ các toán hạng có độ ưu tiên thấp hơn.
 - Chúng ta thực hiện các toán tử có độ ưu tiên cao hơn trước, các toán tử có độ ưu tiên thấp hơn được thực hiện sau.
- Các toán tử có cùng mức ưu tiên có thể tạo thành chuỗi toán tử và được thực hiện theo thứ tự được quy định.
- Thứ tự thực hiện các toán hạng chỉ được quy định cho các toán tử: gọi hàm (), &&, ||, ?:, và toán tử , (*sẽ học sau*).
Thứ tự thực hiện các toán hạng cho các toán tử còn lại là không xác định (phụ thuộc vào triển khai).

Thứ tự ưu tiên và chiều thực hiện chuỗi toán tử

Các toán tử đã học được tổng hợp trong bảng. Các toán tử trong cùng 1 dòng có cùng thứ tự ưu tiên. Các toán tử ở dòng trên có thứ tự cao hơn các toán tử ở dòng dưới.

Ưu tiên cao (thực hiện trước)	Tên toán tử	Ký hiệu	Thứ tự
	Tăng 1 (hậu tố) Giảm 1 (hậu tố)	++ (x++) -- (x--)	<i>Trong phạm vi hiện tại hiếm khi cần quan tâm (thường chỉ áp dụng 1 lần cho 1 toán hạng)</i>
	Tăng 1 (tiền tố) Giảm 1 (tiền tố) Toán tử dấu (tiền tố) Toán tử sizeof	++ (++x) -- (--x) +, - (+x, -x) sizeof (sizeof(int))	
	Nhân Chia Phần dư	* (x * y) / (x / y) % (x / y)	
	Cộng Trừ	+ (x + y) - (x - y)	
	Dịch sang trái Dịch sang phải	<< (x << y) >> (x >> y)	Trái -> Phải
	AND theo bit	& (x & y)	
	XOR theo bit	^ (x ^ y)	Trái -> Phải
	OR theo bit	(x y)	Trái -> Phải
	Gán đơn giản Gán kết hợp	= *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, =	Phải -> Trái
Ưu tiên thấp (thực hiện sau)			

Nội dung

- Biểu thức
 - Nhập và xuất theo định dạng
- 

Ví dụ 4-1. Xuất kết quả tính biểu thức

```
vd4-1.c x
1  #include <stdio.h>
2
3  int main() {
4      int a = 11, b = 3, c = -5;
5      int n, d;
6      n = a / b;
7      d = a - n * b;
8
9      // %d là đặc tả xuất số nguyên có dấu
10     printf("n = %d d = %d (%d)\n",
11            n, d, a % b);
12     printf("%d/%d = %d  %d %% %d = %d\n",
13            a, c, a/c, a, c, a % c);
14
15     // %u - số nguyên không dấu
16     unsigned char ch = 1;
17     printf("%u\n", ch << 10);
18
19     // %f - float hoặc double
20     double d1 = 11, d2 = 3;
21     printf("%f / %f = %f\n", d1, d2, d1 / d2);
22     return 0;
23 }
```

bangoc:\$gcc -o prog vd4-1.c
bangoc:\$./prog
n = 3 d = 2 (2)
11/-5 = -2 11 % -5 = 1
1024
11.000000 / 3.000000 = 3.666667

Bộ cục xuất là thông điệp bao gồm các đặc tả xuất và các ký tự thường.

Các đặc tả xuất sẽ được thay thế bởi giá trị của các biểu thức.

Các ký tự bên ngoài các đặc tả được giữ nguyên khi xuất

Nhập và xuất theo định dạng

Thư viện: <stdio.h>, hàm printf

- Có thể được sử dụng để xuất dữ liệu theo định dạng ra màn hình (luồng xuất tiêu chuẩn/stdout)
- Nguyên mẫu giản lược:

```
int printf (const char *format, ...);
```
- Các tham số:
 - format - là bố cục xuất;
 - ... - Các giá trị được xuất theo bố cục xuất (không bắt buộc).
- Giá trị trả về: Số lượng ký tự đã được in ra hoặc một số âm nếu có lỗi xuất dữ liệu.
- Ví dụ:

```
printf("%d %.3f\n", 10, 2.135);
```

 \\ Trả về 9 nếu thành công

Nhập và xuất theo định dạng₍₂₎

Thư viện: <stdio.h>, hàm scanf

- Có thể được sử dụng để nhập dữ liệu theo định dạng từ bàn phím (luồng nhập tiêu chuẩn/stdin)
- Nguyên mẫu giản lược:

```
int scanf (const char *format, ...);
```
- Các tham số:
 - format - là bố cục nhập;
 - ... - Các con trỏ tới nơi lưu giá trị nhập (Không bắt buộc).
- Giá trị trả về: Trả về số lượng giá trị đã đọc được hoặc EOF trong trường hợp chưa xử lý bất kỳ ký tự nhập nào.
- Ví dụ:

```
if (scanf("%d%d", &a, &b) != 2) {  
    /* Lỗi nhập dữ liệu */  
}
```

Cấu trúc của đặc tả xuất

(Đặc tả xuất/*Output conversion specification*; Chuỗi bố cục xuất/*Template string* = Chuỗi định dạng xuất/*Format string*)

- Cấu trúc khái quát của đặc tả xuất:

% [stt-tham số \$] các cờ **chiều rộng** [. độ chính xác] định kiểu hoặc

% [stt-tham số \$] các cờ **chiều rộng** . * [stt-tham số \$] định kiểu

- Chỉ có mô tả định kiểu là bắt buộc, các mô tả còn lại có thể có hoặc không. Sơ lược ý nghĩa của các thành phần:
 - stt-tham số \$ - Cho biết tham số được sử dụng;
 - các cờ - Điều chỉnh hình thức xuất giá trị;
 - chiều rộng - Điều chỉnh số lượng ký tự tối thiểu được sử dụng;
 - độ chính xác - Số lượng chữ số tối thiểu đối với số nguyên/ hoặc số chữ số phần thập phân (sau làm tròn) đối với số thực.
 - các ký tự hiệu chỉnh kiểu (đứng trước ký tự định kiểu, nếu có)
 - ký tự định kiểu (là thành phần bắt buộc có trong đặc tả).

Ví dụ 4-2. Đặc tả xuất ở dạng đầy đủ

"%1\$+-10.5d|%2\$*3\$.*4\$f"

Ý nghĩa và các thành phần của đặc tả xuất

%1\$+-10.5d

1\$ - Xuất giá trị của tham số đầu tiên;

+- - Các cờ xuất: '+': dấu + cho số dương, '-': Căn lề trái;

10 - Chiều rộng trường xuất = 10 ký tự;

.5 - Độ chính xác (số chữ số tối thiểu, thêm 0 nếu cần);

d - Ký tự định kiểu (số nguyên có dấu).

%2\$*3\$.*4\$f

2\$ - Xuất giá trị của tham số thứ 2;

*3\$ - Chiều rộng trường xuất bằng giá trị tham số thứ 3;

.*4\$ - Độ chính xác bằng giá trị tham số thứ 4;

f - Ký tự định kiểu (số thực ở dạng thông thường).

Nếu không mô tả stt, các tham số được mặc định tuân tự theo các vị trí. Chúng ta chủ yếu sẽ sử dụng các thứ tự mặc định.

Ví dụ 4-2. Mã nguồn minh họa đặc tả xuất

```
6  #include <stdio.h>
7  int main() {
8      double x = 3.1415;
9      int i = 101;
10     printf("1234567890|12345678901234567890\n");
11     // Sử dụng stt tham số (bắt đầu từ 1)
12     printf("%1$+-10.5d|2$*3$.*4$f\n", i, x, 15, 5)
13
14     // Tham số mặc định theo thứ tự tuần tự
15     printf("%+-10.5d|%*.*f\n", i, 15, 5, x);
16
17     // Độ rộng và độ chính xác cố định
18     printf("%+-10.5d|%15.5f\n", i, x);
19     return 0;
20 }
```

Các câu lệnh xuất trên các dòng 12, 15, 18 cho các kết quả tương đương.

```
bangoc:$gcc -o prog vd4-2.c
bangoc:$./prog
1234567890|12345678901234567890
+00101      |          3.14150
+00101      |          3.14150
+00101      |          3.14150
```

Chiều rộng 10 ký tự,
Căn lề trái

Chiều rộng 15 ký
tự, Căn lề phải

Có thể tham khảo thêm các mô tả chi tiết trong tài liệu tham khảo. Tiếp theo chúng ta sẽ tiếp tục phân tích một số đặc tả xuất thường gặp.

Một số cờ định dạng xuất

- ‘-’: Căn lề trái thay vì căn lề phải như được mặc định.
- ‘+’: Xuất dấu + cho số không âm và dấu - cho số âm (mặc định không hiển thị dấu cho số dương)
- ‘#’: Luôn hiển thị dấu thập phân (.) đối với số thực dấu chấm động; Đối với %g và %G còn hiển thị chữ số 0 ở cuối; Đối với %o - luôn hiển thị chữ số 0 ở đầu; Đối với %x và %X luôn hiển thị các tiền tố 0x và 0X.
- ‘0’: Căn chỉnh bằng các chữ số 0 (không làm thay đổi giá trị số đọc được), cờ này bị bỏ qua nếu sử dụng cùng với cờ ‘-’ (căn lề trái).

Một số đặc tả xuất thông thường

Trong trường hợp thông thường đặc tả xuất có thể chỉ bao gồm dấu % và ký tự định kiểu (có thể có các ký tự hiệu chỉnh), không có các mô tả chi tiết. Một số đặc tả xuất thông thường với các kiểu đã học:

Đặc tả	Ý nghĩa
%d, %i	Xuất số nguyên như số nguyên có dấu ở HCS 10
%o	Xuất số nguyên như số nguyên không dấu ở HCS 8
%u	Xuất số nguyên như số nguyên không dấu ở HCS 10
%x, %X	Xuất số nguyên như số nguyên không dấu ở HCS 16.
%f	Xuất số thực dấu chấm động theo hình thức phổ thông.
%e, %E	Xuất số thực dấu chấm động theo hình thức khoa học (có phần mũ).
%g, %G	Xuất số thực dấu chấm động theo hình thức phổ thông hoặc hình thức khoa học tùy theo giá trị.
%c	Xuất một ký tự, tham số được ép kiểu thành unsigned char
%%	Xuất chính ký tự %

Các mô tả chi tiết có thể được thêm vào tùy theo tình huống ứng dụng, ví dụ khi cần xuất các giá trị theo định dạng bảng, v.v.

Các hiệu chỉnh kiểu

Với %d và %i tham số được coi như có kiểu **int**; với %o, %u, %x, và %X tham số được coi như có kiểu **unsigned**. Nếu kiểu đúng của tham số khác với mặc định thì có thể bổ xung các ký tự hiệu chỉnh:

Hiệu chỉnh	Định kiểu: d hoặc i	u, o, x, hoặc X
hh (C99)	signed char	unsigned char
h	short	unsigned short
l	long	unsigned long
L, ll, q	long long	unsigned long long
z	Kiểu có dấu của size_t	Kiểu không dấu của size_t

Với %f, %e, %E, %g, %G tham số được coi như có kiểu double. Có thể bổ xung ký tự (hiệu chỉnh) L => long double.

- Ví dụ hiệu chỉnh định kiểu

%ld - Xuất giá trị thuộc kiểu **long** trong HCS 10.

%Lf - Xuất giá trị thuộc kiểu **long double** trong HCS 10.

Lưu ý: Các tham số còn được ép kiểu tự động (char, short => int; float => double, ...)

Ví dụ 4.3. Một số đặc tả suất đơn giản

```
5 #include <stdio.h>
6
7 int main() {
8     printf("12345678901234567890\n");
9
10    // Xuất số nguyên
11    int i = 1001;
12    long long ll = 123451234512l;
13    printf("%d\n", i); // Xuất thường
14    printf("%15d\n", i); // Căn lề phải
15    printf("%-15d\n", i); // Căn lề trái
16    printf("%15d\n", ll); // NOK - Sai kiểu
17    printf("%15lld\n", ll); // Ok
18    printf("%015d\n", i); // Lấp đầy với 0
19    printf("%-15.5d\n", i); // Thêm 0 nếu cần
20    unsigned char uc = 255;
21    printf("%15u\n", uc + 100); // unsigned
22    printf("%15hhu\n", uc + 100); // unsigned char
23
24    // Xuất số thực
25    double d = 3.141592653589793;
26    long double ld = 3.141592653589793238L;
27    printf("%.3f\n", d); // Làm tròn đến 3 chữ số
28    printf("%15.3f\n", d); // Căn lề phải
29    printf("%-15.f\n", d); // Căn lề trái
30    printf("%.18f\n", ld); // NOK - Sai kiểu
31    printf("%20.18Lf\n", ld); // Ok
32
33    // Xuất ký tự
34    printf("%c%c%c%c%c\n", 'h', 101, 'l', 'l', 'o');
35    return 0;
36 }
```

Độ rộng = 20 ký tự

```
12345678901234567890
1001
1001
1001
-1102817072
123451234512
0000000000001001
01001
355
99
3.142
3.142
3
0.00000000000000000000
3.141592653589793238
```

Tham số của %c được ép kiểu thành unsigned char

Nhập theo định dạng

Sơ lược về scanf

- Cách sử dụng hàm scanf có nhiều điểm tương đồng với hàm printf nhưng không tuyệt đối.
- Một số đặc tả nhập của scanf có thể bỏ qua một số lượng không giới hạn các ký tự khoảng trắng (dấu cách, dấu tab, dấu xuống dòng, v.v., tham khảo hàm isspace).
 - Thiết kế này giúp cho việc đọc các giá trị có phần dễ hơn.
- Một sự khác biệt cần lưu ý khác (do mục đích nhập, hàm scanf phải lưu giá trị đọc được vào các biến), các tham số truyền cho scanf phải là ***các con trỏ*** tới các biến.
 - Hàm scanf lưu các giá trị vào các đối tượng được trỏ tới bởi các tham số.

Chuỗi bố cục nhập

- Bao gồm các đặc tả nhập (bắt đầu với %) cùng với các ký tự thông thường.
- Khi xử lý một dấu cách (hoặc bất kỳ ký tự trắng nào khác, tham khảo isspace) trong bố cục nhập, scanf bỏ qua một số lượng bất kỳ các dấu khoảng trắng liên tiếp trong luồng ký tự nhập.
- Các ký tự khác (không phải khoảng trắng, và không phải là thành phần của đặc tả nhập) phải được khớp chính xác với ký tự trong luồng nhập, hoặc sẽ phát sinh lỗi nhập.
 - *(Trong tình huống phát sinh lỗi, hàm scanf trả về ngay lập tức, và gần gọi hàm tiếp theo bắt đầu đọc từ vị trí ký tự lỗi).*
- Đặc tả nhập mô tả giá trị cần được đọc.

Cấu trúc tổng quát của đặc tả nhập

- Đặc tả nhập có thể bao gồm các thành phần:
 - % các cờ **chiều rộng** định kiểu
- *(Không có thành phần độ chính xác trong đặc tả nhập).*
- Cờ là thành phần không bắt buộc:
 - ‘*’ - Khi xử lý đặc tả nhập có dấu ‘*’ scanf đọc giá trị (theo phần còn lại của đặc tả) và bỏ qua giá trị (không lưu).
- Chiều rộng trường nhập là thành phần không bắt buộc:
 - Là số nguyên - Khi xử lý đặc tả nhập có mô tả chiều rộng, tiến trình đọc giá trị dừng lại khi đã hoàn thành đọc giá trị (gặp một ký tự không hợp lệ) hoặc đã đọc hết chiều rộng trường nhập (số lượng ký tự đã đọc == chiều rộng).
- Các ký tự hiệu chỉnh kiểu là thành phần không bắt buộc.
- Ký tự định kiểu là thành phần bắt buộc.

Một số đặc tả nhập thông thường

Trong trường hợp thông thường, đặc tả nhập có thể chỉ bao gồm dấu % và ký tự định kiểu (có thể có thêm các ký tự hiệu chỉnh ở phía trước). Một số ký đặc tả cho các kiểu số đã học:

Đặc tả nhập	Ý nghĩa
%d	Đọc 1 số nguyên có dấu trong HCS 10.
%i	Đọc 1 số nguyên được viết theo bất kỳ định dạng nào trong C.
%o	Đọc 1 số nguyên không dấu được viết ở HCS 8.
%u	Đọc 1 số nguyên không dấu được viết ở HCS 10.
%x, %X	Đọc 1 số nguyên không dấu được viết ở HCS 16.
%e, %f, %g, %E, %G	Đọc 1 số thực dấu chấm động.
%c	Đọc 1 hoặc nhiều ký tự (được mô tả bởi chiều rộng).
%n	Không đọc ký tự nào, chỉ lưu số lượng ký tự đã đọc vào tham số.
%%	Khớp với ký tự % trong luồng đọc.

*Cách viết các hằng giá trị đã được mô tả trong chương 3, nhưng lưu ý không sử dụng các hậu tố khi nhập. Tham khảo thêm các hàm **strtol** và **strtod**.*

Các hiệu chỉnh kiểu

Các đặc tả nhập số nguyên mặc định tham số có kiểu `int *` hoặc `unsigned int *`. Trong trường hợp tham số lưu giá trị nhập có kiểu khác, chúng ta có thể bổ xung các hiệu chỉnh kiểu:

Hiệu chỉnh	Định kiểu: d hoặc i	u, o, x, hoặc X
hh (C99)	<code>signed char *</code>	<code>unsigned char *</code>
h	<code>short *</code>	<code>unsigned short *</code>
l	<code>long *</code>	<code>unsigned long *</code>
ll, L, q	<code>long long *</code>	<code>unsigned long long *</code>
z (C99)	<code>size_t *</code>	<code>size_t *</code>

Các đặc tả số thực dấu chấm động mặc định tham số có kiểu `float *`. Trong các trường hợp khác có thể bổ xung: 'l' nếu tham số có kiểu `double *`, hoặc 'L' nếu tham số có kiểu `long double *`.

Ví dụ hiệu chỉnh kiểu:

- `%lu` - Đọc giá trị kiểu `unsigned long`;
- `%lf` - Đọc giá trị kiểu `double`.

Ví dụ 4.4. Nhập giá trị thông thường

```
5  #include <stdio.h>
6
7  int main() {
8      long l;
9      double d;
10     char c;
11     scanf("%ld%lf %c", &l, &d, &c);
12     printf("Bạn đã nhập: \n"
13           "l = %ld\n"
14           "d = %f\n"
15           "c = %c\n", l, d, c);
16     int dd, mm, y;
17     printf("Nhập một ngày theo định dạng d/m/y: ");
18     scanf("%d / %d / %d", &dd, &mm, &y);
19     printf("Ngày %d Tháng %d Năm %d\n", dd, mm, y);
20     return 0;
21 }
```

11 12.123 a
Bạn đã nhập:
l = 11
d = 12.123000
c = a
Nhập một ngày theo định dạng d/m/y:
3/11 /2021
Ngày 3 Tháng 11 Năm 2021

Ví dụ 4.5. Sử dụng cờ và chiều rộng

```
6  #include <stdio.h>
7  int main() {
8      int i;
9      double d;
10     unsigned int n;
11     scanf("%3d%*3d%5lf%n", &i, &d, &n);
12     printf("Bạn đã nhập: \n"
13           "i = %d\n"
14           "d = %f\n"
15           "Đã đọc %u ký tự.\n", i, d, n);
16     return 0;
17 }
```

Bỏ qua

1235671.123
Bạn đã nhập:
i = 123
d = 1.123000
Đã đọc 11 ký tự.

Lưu ý: Với kiểu double, chúng ta sử dụng đặc tả %lf để đọc nhưng có thể xuất với đặc tả %f.

*Các tham số cho hàm scanf là các con trỏ (có dấu & trước tên biến)
Các chi tiết về con trỏ sẽ được cung cấp sau.*

