Ngôn ngữ lập trình C

Bài 4. Nhập và xuất theo định dạng, Biểu thức lô-gic, và Các câu lệnh

Soạn bởi: TS. Nguyễn Bá Ngọc

2022

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- Các câu lệnh
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

Nội dung

- **Nhập và xuất theo định dạng**
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- Các câu lệnh
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

Xuất theo định dạng

Hàm printf, thư viện <stdio.h>

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

```
int printf (const char *format, ...);
```

- Các tham số:
  - format - là bố cục xuất, có kiểu chuỗi ký tự (*học sau*);
  - ... - Danh sách các giá trị được đưa vào bố cục xuất (nếu cần).
- Giá trị trả về: Số lượng ký tự đã được in ra hoặc một số âm nếu có lỗi xuất dữ liệu.

Sử dụng hàm printf

- Cú pháp gọi hàm:
  - printf("Bố-cục-xuất", …);
- Bố-cục-xuất có vai trò như mẫu nội dung, là đoạn văn bản có thể chứa các đặc tả đánh dấu các vị trí xuất và mô tả định dạng xuất của các giá trị được cung cấp sau đó.
- Ví dụ:
  - printf("Tổng = %d, Hiệu = %d", a + b, a - b);
    - Bố cục xuất: "Tổng = %d, Hiệu = %d" chứa 2 đặc tả
    - %d là đặc tả xuất số nguyên kiểu int ở HCS 10 với định dạng mặc định.
    - Giá trị biểu thức a + b được thay thế vào vị trí %d thứ nhất, giá trị a - b được thay thế vào vị trí %d thứ 2.
  - printf("%d %.3f", 10, 2.135); // Trong trường hợp thành công:
    - Xuất ra: 10 2.135
    - Kết quả gọi hàm = 8

Cấu trúc đặc tả xuất

(Đặc tả xuất/Output conversion specification; Chuỗi mẫu xuất/Output template string; Chuỗi định dạng xuất/Output format string)

- (Các) Cấu trúc tổng quát:

% [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**

- Các thành phần đặc tả:
  - stt-tham-số $ - Lựa chọn đối số theo số thứ tự;
  - các-cờ - Điều chỉnh hình thức xuất: Dấu, căn chỉnh;
  - 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.
  - định-kiểu (có thể có các ký tự hiệu chỉnh được kết hợp với ký tự định kiểu) mô tả kiểu định dạng xuất được áp dụng. Định kiểu là thành phần bắt buộc trong đặc tả.

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)

‘ ‘: Nếu kết quả xuất theo %d, %i không bắt đầu với dấu thì thêm 1 khoảng trắng (bỏ qua nếu kết hợp với +)

‘#’: 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), bị bỏ qua nếu sử dụng cùng với cờ ‘-’.

Các ký tự định kiểu

Đặc tả	Kiểu đối số	Ý nghĩa
%d, %i	int	Xuất số nguyên có dấu ở HCS 10
%u	unsigned	Xuất số nguyên không dấu ở HCS 10
%o		Xuất số nguyên không dấu ở HCS 8
%x, %X		Xuất số nguyên không dấu ở HCS 16
%f	double	Xuất số thực theo định dạng thường
%e, %E		Xuất số thực theo định dạng khoa học
%g, %G		Xuất số thực theo định dạng thích hợp nhất với giá trị được xuất.

*Lưu ý trong gọi hàm printf:

+ Các đối số kiểu số nguyên với hạng < rank(int) được tự động ép kiểu int hoặc unsigned (quy tắc nâng hạng).

+ Các đối số kiểu float được tự động ép kiểu double.

Có thể kết hợp các ký tự hiệu chỉnh với các ký tự định kiểu.

Các hiệu chỉnh kiểu

Các hiệu chỉnh kiểu có thể được thêm vào để thông báo cho printf *biết kiểu đúng của đối số khi cần:*

Hiệu chỉnh	**Kết hợp với : d hoặc i**	**u, o, x, hoặc X**
hh (C99)	signed char	unsigned char
h	short	unsigned short
l	long	unsigned long
ll, L (GNU)	long long	unsigned long long
z	Kiểu có dấu của size_t	Kiểu không dấu của size_t

- Để xuất số nguyên các hiệu chỉnh hh và h không thực sự cần thiết, bởi vì giá trị kiểu char và short được tự động ép kiểu int …
  - Hiệu chỉnh hh yêu cầu ép kiểu ngược lại thành char
  - h yêu cầu ép kiểu ngược lại thành short
- Để xuất số thực kiểu long double, ký tự hiệu chỉnh L được sử dụng kết hợp với f, e, E, g, hoặc G
  - Giá trị kiểu float được tự động ép kiểu double

Lựa chọn định kiểu xuất theo kiểu đối số

Kiểu	**Định kiểu xuất**
Có dấu (thường là mã bù-2)	
signed char	%d, %i, %hhd, %hhi
short	%d, %i, %hd, %hi
int	%d, %i
long	%ld, %li
long long	%lld, %lli
Không dấu (thuần nhị phân)	
unsigned char	%u, %o, %x, %X, %hhu, …
unsigned short	%u, %o, %x, %X, %hu, …
unsigned	%u, %o, %x, %X
unsigned long	%lu, %lo, %lx, %lx
unsigned long long	%llu, %llo, %llx, %llX
Số thực dấu chấm động	
float	%f, %e, %E, %g, %G
double	%f, %e, %E, %g, %G
long double	%Lf, %Le, %LE, %Lg, %LG

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

```
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
```

```c
#include <stdio.h>

int main() {
  int a = 11, b = 3, c = -5;
  int n, d;
  n = a / b;
  d = a - n * b;

  // %d là đặc tả xuất số nguyên có dấu
  printf("n = %d d = %d (%d)\n",
                 n,    d, a % b);
  printf("%d/%d = %d   %d %% %d = %d\n",
            a, c,  a/c,  a,    c, a % c);

  // %u - số nguyên không dấu
  unsigned char ch = 1;
  printf("%u\n", ch << 10);

  // %f - float hoặc double
  double d1 = 11, d2 = 3;
  printf("%f / %f = %f\n", d1, d2, d1 / d2);
  return 0;
}
```

Bố cục xuất là mẫu thông báo 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

Đặc tả xuất được bắt đầu với dấu %

Ví dụ 4.2a. Các đặc tả xuất đơn giản

%d - Xuất số nguyên kiểu int

%10d - Sử dụng độ rộng = 10, căn lề phải

%-10d - Căn lề trái

%f - Xuất số thực kiểu double

%.3f - Sử dụng độ chính xác = 3 chữ số

%10.3f - Độ rộng = 10, căn lề phải

%-10.3f - Căn lề trái

…

Lưu ý: Sử dụng định kiểu đúng với kiểu của đối số. Nếu kiểu đối số không đúng với đặc tả thì hành vi là bất định.

Ví dụ 4.2b. Một số đặc tả suất đơn giản

```c
#include <stdio.h>

int main() {
  printf("12345678901234567890\n");

  // Xuất số nguyên
  int i = 1001;
  long long ll = 1234512345121;
  printf("%d\n", i);  // Xuất thường
  printf("%15d\n", i);  // Căn lề phải
  printf("%-15d\n", i);  // Căn lề trái
  printf("%15d\n", ll);  // NOK - Sai kiểu
  printf("%15lld\n", ll);  // Ok
  printf("%015d\n", i);  // Lấp đầy với 0
  printf("%-15.5d\n", i);  // Thêm 0 nếu cần
  unsigned char uc = 255;
  printf("%15u\n", uc + 100);  // unsigned
  printf("%15hhu\n", uc + 100);  // unsigned char

  // Xuất số thực
  double d = 3.141592653589793;
  long double ld = 3.14159265358979323L;
  printf("%.3f\n", d);  // Làm tròn đến 3 chữ số
  printf("%15.3f\n", d);  // Căn lề phải
  printf("%-15.f\n", d);  // Căn lề trái
  printf("%.18f\n", ld);  // NOK -  Sai kiểu
  printf("%20.18Lf\n", ld);  // Ok

  // Xuất ký tự
  printf("%c%c%c%c%c\n", 'h', 101, 'l', 'l', 'o');
  return 0;
}
```

Độ rộng = 20 ký tự

```
12345678901234567890
1001
                1001
1001
     -1102817072
   123451234512
00000000001001
01001
                 355
                  99
3.142
              3.142
3
0.00000000000000000
3.14159265358979323
```

Tham số của %c được ép kiểu thành unsigned char

Ví dụ 4-3a. Đặc tả xuất với đầy đủ thành phần

```
"%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 tuần tự theo các vị trí. Chúng ta chủ yếu sẽ sử dụng các thứ tự mặc định.

Ví dụ 4-3b. 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  }
```

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

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

Chiều rộng 10 ký tự, căn lề trái

Chiều rộng 15 ký tự, căn lề phải

Nội dung

- **Nhập và xuất theo định dạng**
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- Các câu lệnh
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

Nhập theo định dạng

Hàm scanf, thư viện: <stdio.h>

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

```
int scanf (const char *format, ...);
```

- Các tham số:
  - format - là chuỗi bố cục nhập;
  - … - Các con trỏ tới các đối tượng lưu giá trị nhập theo các đặc tả trong bố cục nhập (nếu cần).
- Giá trị trả về: Số lượng giá trị đã đọc được hoặc EOF trong trường hợp chưa xử lý bất kỳ ký tự nhập nào.

Sử dụng hàm scanf

- Định dạng gọi hàm
  - scanf("bố-cục-nhập", …)
- Chuỗi bố-cục-nhập mô tả cách chuyển chuỗi ký tự nhập thành các giá trị nhập
- Ví dụ:
  - scanf("%d%d", &a, &b)
    - "%d%d" là chuỗi bố cục nhập chứa 2 đặc tả nhập các số nguyên
    - &a, &b là các con trỏ tới các đối tượng a, b (phải có kiểu int trong trường hợp này).
    - Nếu người dùng nhập từ bàn phím 10 20 rồi nhấn phím Enter, thì (t*rong trường hợp thành công*) …
    - … a = 10, b = 20. Kết quả gọi hàm = 2.

Cấu trúc đặc tả nhập

- Cấu trúc tổng quát:

  `% *các-cờ* **chiều-rộng** định-kiểu`

- Các thành phần:
  - Cờ: Khi xử lý đặc tả nhập có dấu ‘*" scanf đọc giá trị nhưng bỏ qua (không lưu).
  - Chiều-rộng trường nhập: 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).
  - Định-kiểu (các ký tự hiệu chỉnh có thể được kết hợp với ký tự định kiểu) mô tả kiểu định dạng được áp dụng.
  - *(Định-kiểu là thành phần bắt buộc)*

Các ký tự định kiểu nhập

Đặc tả nhập	**Kiểu dữ liệu (của giá trị)**	**Ý nghĩa**
%d	int	Đọc 1 số nguyên có dấu trong HCS 10.
%i		Đọc 1 số nguyên được viết theo bất kỳ định dạng nào trong C.
%u	unsigned	Đọc 1 số nguyên không dấu được viết ở HCS 10.
%o		Đọc 1 số nguyên không dấu được viết ở HCS 8.
%x, %X		Đọc 1 số nguyên không dấu được viết ở HCS 16.
%f, %e, %E, %g, %G	float	Đọc 1 số thực dấu chấm động.
%n	int	Lưu số lượng ký tự đã đọc vào đối số (không đọc ký tự nào khi xử lý %n).

**Lưu ý: Đối số là con trỏ, để nhập giá trị có kiểu* int *thì đối số phải có kiểu* int*

Tham khảo thêm các hàm **strtol, strtoul** *và* **strtod**.

Các hiệu chỉnh kiểu

Các hiệu chỉnh kiểu có thể được thêm vào để thông báo cho scanf *biết kiểu đúng của đối tượng lưu giá trị nhập khi cần:*

Hiệu chỉnh	**Kết hợp với: d hoặc i**	**u, o, x, hoặc X**
hh (C99)	signed char	unsigned char
h	short	unsigned short
l	long	unsigned long
ll, L (GNU)	long long	unsigned long long
z (C99)	size_t	size_t

- Trong trường hợp nhập số thực (với các định kiểu f, e, E, g, hoặc G) có thể bổ xung các hiệu chỉnh:
  - l (ví dụ %lf): Giá trị có kiểu double
  - L (ví dụ %Lf): Giá trị có kiểu long double

Lựa chọn định kiểu theo kiểu giá trị

Kiểu (của giá trị)	**Định kiểu nhập**
Có dấu (thường là mã bù-2)	
signed char	%hhd, %hhi
short	%hd, %hi
int	%d, %i
long	%ld, %li
long long	%lld, %lli
Không dấu (thuần nhị phân)	
unsigned char	%hhu, %hho, %hhx, %hhX
unsigned short	%hu, %ho, %hx, %hX
unsigned	%u, %o, %x, %X
unsigned long	%lu, %lo, %lx, %lx
unsigned long long	%llu, %llo, %llx, %llX
Số thực dấu chấm động	
float	%f, %e, %E, %g, %G
double	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG

So sánh scanf và printf

- Các hàm scanf và printf được xây dựng với cấu trúc tương tự, điển hình như có 1 số định kiểu có thể được dùng trong cả 2 hàm để nhập và xuất. Tuy nhiên trong mỗi khái niệm vẫn tồn tại những khác biệt tương đối lớn:
- Đối với chuỗi bố cục nhập:
  - Khi xử lý 1 dấu khoảng trắng (dấu cách, dấu tab, dấu xuống dòng, v.v., tham khảo hàm isspace) scanf bỏ qua tất cả dấu khoảng trắng liên tiếp trong luồng ký tự nhập.
    - *(Thiết kế này giúp cho việc nhập các giá trị số có phần dễ hơn).*
  - 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.
- Các đối số tương ứng với các đặc-tả-nhập trong scanf phải là ***các con trỏ*** (do hàm scanf phải lưu các giá trị).

Ví dụ 4.4. Định dạng nhập đơn giản

```c
#include <stdio.h>

int main() {
  long l;
  double d;
  char c;
  scanf("%ld%lf %c", &l, &d, &c);
  printf("Bạn đã nhập: \n"
         "l = %ld\n"
         "d = %f\n"
         "c = %c\n", l, d, c);
  int dd, mm, y;
  printf("Nhập một ngày theo định dạng d/m/y: ");
  scanf("%d / %d / %d", &dd, &mm, &y);
  printf("Ngày %d Tháng %d Năm %d\n", dd, mm, y);
  return 0;
}
```

```
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. Đặc tả nhập với 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

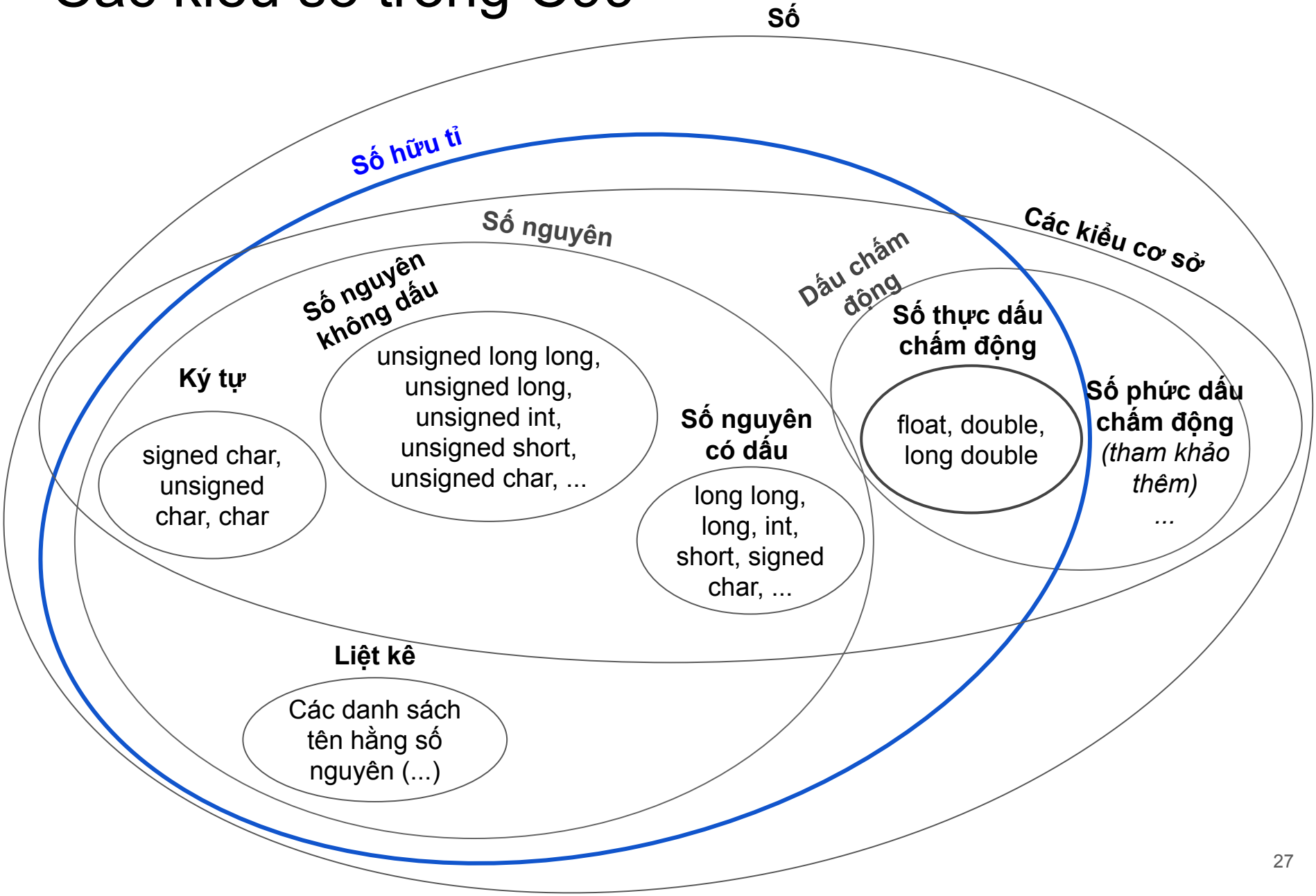
```
1235671.123
Bạn đã nhập:
i = 123
d = 1.123000
Đã đọc 11 ký tự.
```

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 trong ví dụ)*

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- Các câu lệnh
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

Các kiểu số trong C99



Các toán tử thứ tự

- Định dạng:
  - a < b
  - a > b
  - a <= b
  - a >= b
- Điều kiện:
  - Các toán hạng có kiểu số hữu tỉ;
  - *(Trường hợp toán hạng có kiểu con trỏ sẽ được học sau)*
- Ý nghĩa:
  - Cho kết quả = 1 kiểu int nếu đúng, = 0 nếu ngược lại.
  - Quy tắc ép kiểu số được áp dụng.
- Ví dụ:
  - 1 < 2 cho kết quả == 1
  - 2 > 3 cho kết quả == 0

Các toán tử so sánh bằng

- Định dạng:
  - a == b
  - a != b
- Điều kiện:
  - Các toán hạng có kiểu số;
  - *(Trường hợp toán hạng có kiểu con trỏ sẽ được học sau)*
- Ý nghĩa:
  - Cho kết quả = 1 kiểu int nếu đúng, = 0 nếu ngược lại.
  - Với 2 toán hạng bất kỳ, chỉ có duy nhất 1 trong 2 quan hệ (== hoặc !=) cho kết quả đúng.
    - a != b tương đương với !(a == b)
  - Quy tắc ép kiểu số được áp dụng
- Ví dụ:
  - 2 == 3 cho kết quả sai
  - 2 != 3 cho kết quả đúng.

Toán tử lô-gic AND

- Định dạng:
  - a && b
- Điều kiện:
  - Các toán hạng có kiểu số;
  - *(Trường hợp toán hạng có kiểu con trỏ sẽ được học sau)*
- Ý nghĩa:
  - Cho giá trị = 1 kiểu int nếu cả 2 toán hạng != 0, = 0 nếu ngược lại
  - **Thứ tự thực hiện các toán hạng** được đảm bảo từ trái sang phải. Toán hạng thứ nhất là 1 đơn vị tuần tự, nếu có giá trị == 0 thì toán hạng thứ 2 không được tính.
- Ví dụ:
  - int x = 10;
  - 2 > 3 && ++x <= 10 cho kết quả = 0, x == 10

Toán tử lô-gic OR

- Định dạng:
  - a || b
- Điều kiện:
  - Các toán hạng có kiểu số.
  - *(Trường hợp toán hạng có kiểu con trỏ sẽ được học sau)*
- Ý nghĩa:
  - Cho kết quả = 1 kiểu int nếu có ít nhất 1 trong 2 toán hạng có giá trị != 0, = 0 nếu ngược lại.
  - Thứ tự thực hiện các toán hạng được đảm bảo từ trái sang phải. Toán hạng thứ nhất là 1 đơn vị tuần tự, nếu có giá trị != 0 thì toán hạng thứ 2 không được tính.
- Ví dụ:
  - int x = 10;
  - 3 < 5 || ++x > 10 Cho kết quả = 1, x = 10

Toán tử phủ định lô-gic

- Định dạng:
  - !a
- Điều kiện:
  - Toán hạng có kiểu số.
  - *(Trường hợp toán hạng có kiểu con trỏ sẽ được học sau)*
- Ý nghĩa:
  - Cho kết quả = 1 kiểu int nếu toán hạng có giá trị == 0, = 0 nếu ngược lại, tương đương với 0 == a.
- Ví dụ:

  int a = 0, b = 1;

  !b Cho kết quả == 0 kiểu int

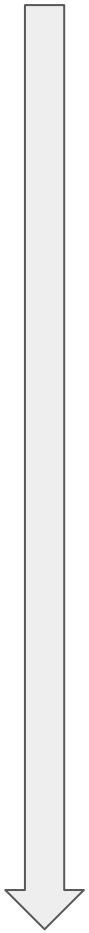
  !(a == b) Cho kết quả == 1 kiểu int (như a != b).

Toán tử lựa chọn

- Định dạng:
  - a? b : c
- Điều kiện:
  - a phải đáp ứng các điều kiện để có thể thực hiện phép so sánh a != 0
  - b và c đều có kiểu số *(hoặc tương đương về kiểu các trường hợp khác, sẽ học sau).*
- Ý nghĩa:
  - a được tính trước tiên, có 1 điểm tuần tự ngay sau đó.
  - b chỉ được tính nếu a != 0 (a đúng), c chỉ được tính nếu ngược lại (a == 0), kết quả là giá trị của toán hạng được tính.
  - Kiểu của biểu thức là kiểu thu được như khi áp dụng quy tắc ép kiểu số cho b và c (trong trường hợp b và c có kiểu số).
- Ví dụ:
  - 2 < 3? 10: 3.5 cho kết quả = 10, kiểu double.

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
Tăng 1 (hậu tố) Giảm 1 (hậu tố)	++ (x++) -- (x--)	*Trong phạm vi hiện tại hiếm khi cần quan tâm (thường chỉ áp dụng 1 lần cho 1 toán hạng)*
Tăng, giảm 1 (tiền tố) Dấu (tiền tố) Phủ định lô-gic sizeof	++, -- (++x, --x) +, - (+x, -x) ! (!e) sizeof (sizeof(int))	
Ép kiểu	() (double)5	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

Ví dụ 4.6. Ép kiểu số trong so sánh

```
int x = -1;

unsigned y = 10;

Biểu thức nào cho kết quả đúng:

x < y

x > y
```

Thử kiểm tra kết quả với chương trình C?

Ví dụ 4.7. So sánh số thực

double x = 0.1, y = 10.1 - 10;

Biểu thức so sánh nào cho kết quả đúng?

x < y

x > y

x == y

Thử kiểm tra kết quả với chương trình C?

Ví dụ 4.8.a. Biểu thức lô-gic AND

```
int a = 1;
int b = 1;
(a % 2 == 0 && ++b > 1)
```

=> Kết quả biểu thức = 0, b = 1 (ngắt sớm)

a = 2;

(**a % 2 == 0** && **++b > 1**)

=> Kết quả biểu thức = 1, b = 2 (++b được thực hiện).

Ví dụ 4.8.b. Biểu thức lô-gic OR

```
int a = 2, b = 1;
(a % 2 == 0 || ++b % 2 == 0)
```

=> Kết quả biểu thức = 1, b = 1 (ngắt sớm).

```
a = 3;
(a % 2 == 0 || ++b % 2 == 0)
```

=> Kết quả biểu thức = 1, b = 2 (++b được thực hiện).

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- **Các câu lệnh**
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

Câu lệnh

- Câu lệnh đặc tả 1 hành động được thực hiện. Các lệnh mặc định được thực hiện tuần tự nếu không có chỉ dẫn khác.
  - Ví dụ: Lệnh goto
- Phân loại câu lệnh trong C:
  - Câu lệnh tính biểu thức
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Câu lệnh rẽ nhánh
  - Câu lệnh lặp
  - Câu lệnh di chuyển

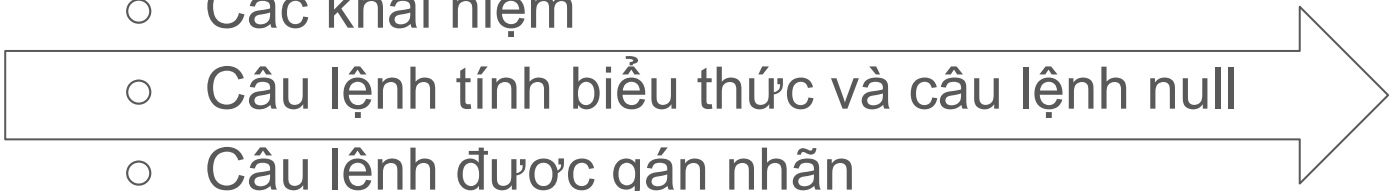
Khối

- Trong C chúng ta có thể kết hợp nhiều mô tả (khai báo, câu lệnh) thành 1 khối.
  - Có phạm vi riêng là tập con của phạm vi chứa nó.
    - Có thể tái sử dụng định danh đã có trong phạm vi lớn hơn.
  - Đồng thời cũng là 1 đơn vị mô tả.
- Các loại câu lệnh được coi là khối trong C:
  - Lệnh gộp / Kết hợp: *Gom các mô tả trong 1 cặp {}*
  - Lệnh rẽ nhánh
  - Lệnh lặp

Kết hợp các mô tả bằng các từ khóa theo quy tắc ngữ pháp.

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- **Các câu lệnh**
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển



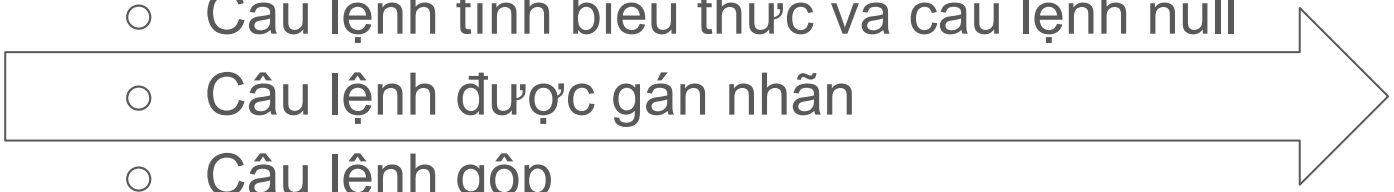
Câu lệnh tính biểu thức và câu lệnh null

- Câu lệnh tính biểu thức chỉ bao gồm biểu thức và kết thúc bằng dấu ;. Ví dụ:
  - a; // Câu lệnh tính biểu thức a
  - a + b; // Câu lệnh tính biểu thức a + b
- Trường hợp đặc biệt: Câu lệnh chỉ bao gồm dấu ; được gọi là câu lệnh null
  - Không làm gì cả nhưng cần thiết cho 1 số tình huống để đảm bảo tính chặt chẽ cú pháp do yêu cầu có câu lệnh.

Biểu thức cũng có thể là thành phần của câu lệnh lớn hơn, ví dụ biểu thức điều khiển trong câu lệnh rẽ nhánh hoặc lặp.

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- **Các câu lệnh**
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

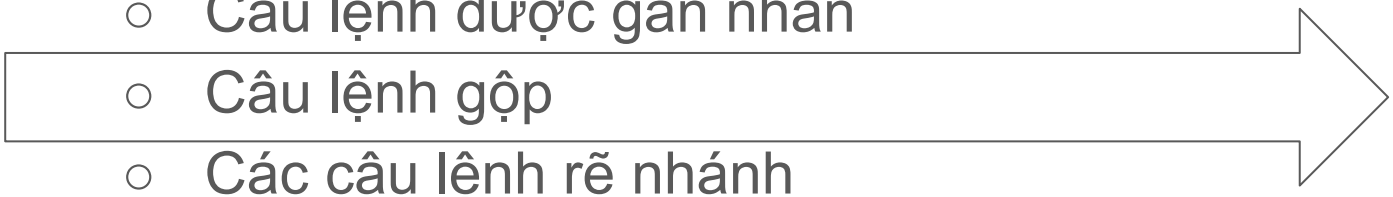


Câu lệnh được gán nhãn

- Định dạng:
  - định-danh: câu-lệnh
  - case hằng-nguyên: câu-lệnh
  - default: câu-lệnh
- Điều kiện:
  - Nhãn case và nhãn default chỉ được sử dụng trong câu lệnh switch. *(thông tin cụ thể hơn sẽ được cung cấp cùng với câu lệnh* switch*)*
- Ý nghĩa:
  - Bất kỳ câu lệnh nào cũng có thể được gán nhãn. Gán nhãn không làm thay đổi luồng / thứ tự thực hiện lệnh.
  - Có thể di chuyển tới câu lệnh có nhãn là định-danh bằng lệnh goto
    - goto định-danh;

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- **Các câu lệnh**
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển



Câu lệnh gộp

- Có thể gom nhiều mô tả liên tục trong 1 cặp dấu {}, phần mã nguồn thu được là 1 câu lệnh gộp.
  - *(tương tự đặt biểu thức trong* () *để tạo biểu thức cơ bản)*
  - Mỗi câu lệnh gộp là 1 khối.
  - Câu lệnh gộp cũng có thể chỉ bao gồm cặp dấu {}
  - Thành phần có thể là câu lệnh gộp, tạo thành nhiều phạm vi lồng nhau.
- Các trường hợp sử dụng tiêu biểu:
  - Câu lệnh gộp có thể là thân hàm: Như thân hàm main.
  - Thân vòng lặp hoặc 1 nhánh của câu lệnh rẽ nhánh *(sẽ học trong bài này)*.
  - Câu lệnh gộp cũng có thể được sử dụng chỉ để tạo phạm vi.

Ví dụ 4.9. Câu lệnh gộp

vd5-1.c

```
1  #include <stdio.h>
2  int main()
3  {
4     int s = 0;
5     {
6        int s = 100;
7        {
8           int s = 202;
9           printf("s = %d\n", s);
10       }
11       printf("s = %d\n", s);
12    }
13    printf("s = %d\n", s);
14    return 0;
15 }
```

S1

S2

S3

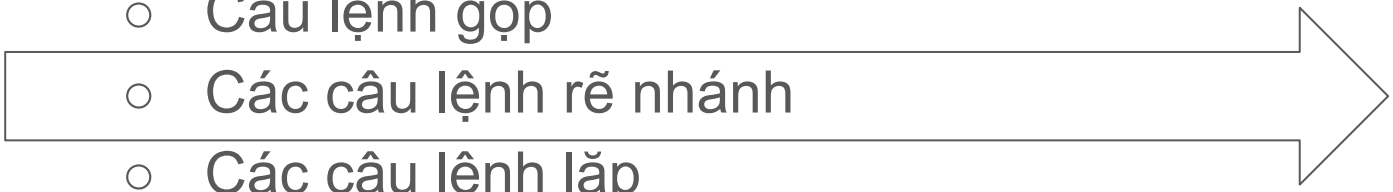
S1, S2, S3 đều là các câu lệnh gộp, trong đó phạm vi S1 là tập con của phạm vi tệp, phạm vi S2 là tập con của S1, phạm vi S3 là tập con của S2

```
s = 202
s = 100
s = 0
```

Cùng 1 định danh s được sử dụng cho 3 đối tượng trong 3 phạm vi (khối) khác nhau.

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- **Các câu lệnh**
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển



Các câu lệnh rẽ nhánh

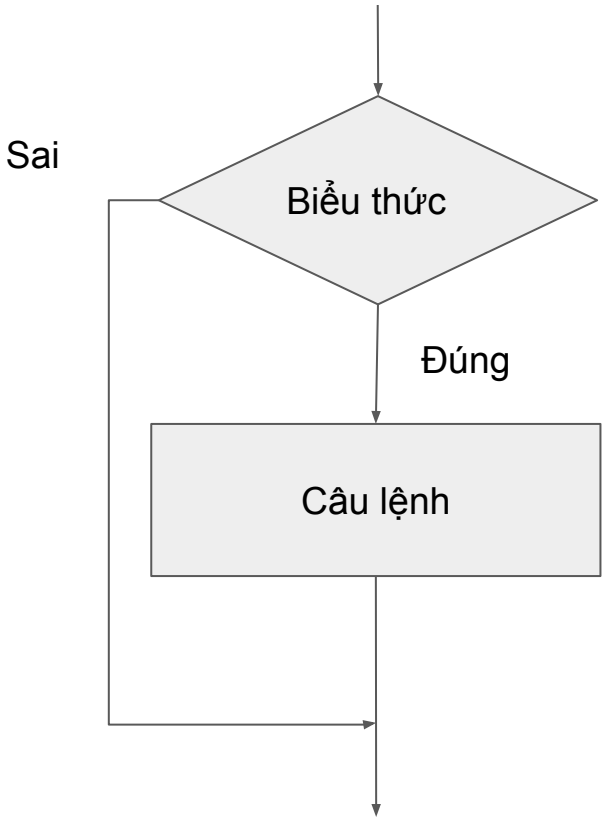
- Câu lệnh rẽ nhánh thực hiện lựa chọn trong 1 tập câu lệnh theo giá trị của biểu thức điều khiển:
- Có 3 định dạng:
  - if (*biểu thức*) *câu lệnh*
  - if (*biểu thức*) *câu lệnh* else *câu lệnh*
  - switch (*biểu thức*) *câu lệnh*
- Câu lệnh rẽ nhánh là **khối** với phạm vi là tập con của phạm vi chứa nó, đồng thời mỗi câu lệnh thành phần cũng là 1 khối với phạm vi là tập con của phạm vi câu lệnh rẽ nhánh.

Câu lệnh if

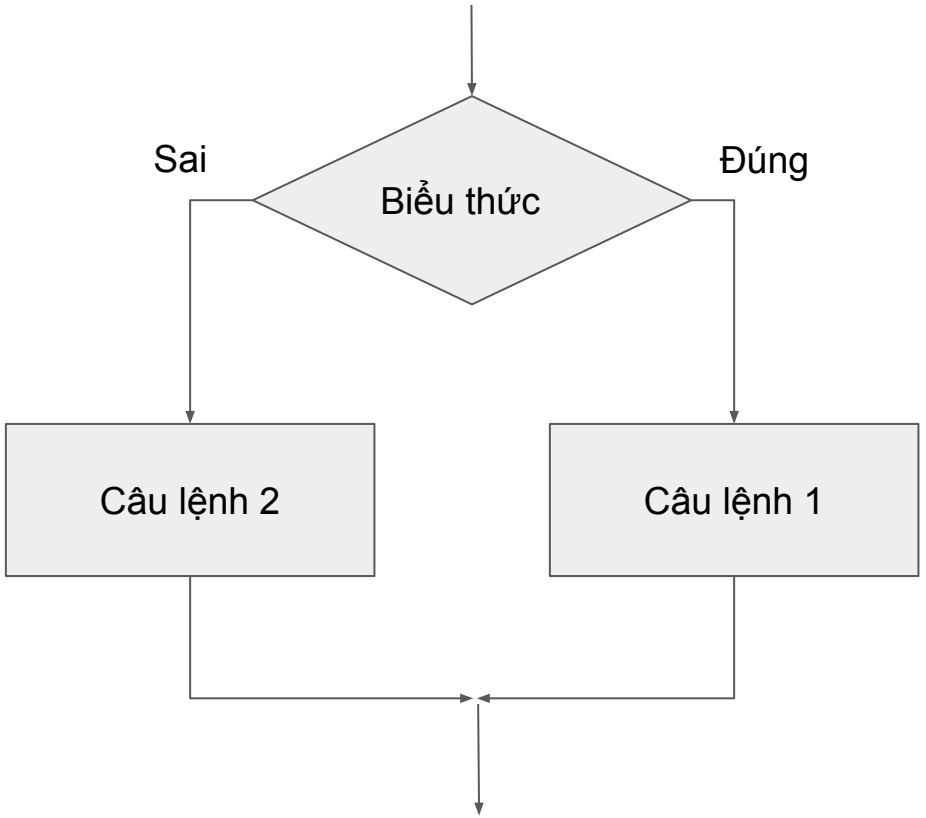
- Định dạng
  - 1 nhánh: if (*biểu thức*) *câu lệnh*
  - 2 nhánh: if (*biểu thức*) *câu lệnh 1* else *câu lệnh 2*
- Điều kiện
  - Biểu thức điều khiển thỏa mãn các điều kiện cần thiết cho phép so sánh !=, (biểu thức) != 0
- Ý nghĩa:
  - if (*biểu thức*) *câu lệnh*: Câu lệnh thành phần chỉ được thực hiện nếu biểu thức điều khiển có giá trị != 0, nếu ngược lại thì câu lệnh thành phần không được thực hiện.
  - if (*biểu thức*) *câu lệnh 1* else *câu lệnh 2*: Câu lệnh 1 chỉ được thực hiện nếu biểu thức điều khiển != 0, câu lệnh 2 chỉ được thực hiện nếu biểu thức điều khiển == 0.

Lưu đồ thực hiện lệnh if

if (biểu thức) câu-lệnh



if (biểu thức) câu-lệnh 1
else câu-lệnh 2



Ví dụ 4.10. Câu lệnh if

```
bangoc:$gcc -o prog vd5-2.c
bangoc:$./prog
Nhập 1 số nguyên dương: -1
-1 không thỏa mãn yêu cầu.
bangoc:$./prog
Nhập 1 số nguyên dương: 5
1 + ... + 5 = 9
bangoc:$./prog
Nhập 1 số nguyên dương: 6
2 + ... + 6 = 12
```

```c
#include <stdio.h>

int main() {
  int n;
  printf("Nhập 1 số nguyên dương: ");
  scanf("%d", &n);
  if (n <= 0 ) {
    printf("%d không thỏa mãn yêu cầu.\n", n);
    return 0;
  }
  if (n % 2 == 0) {
    int sum = n * (n + 2) / 4;
    printf("2 + ... + %d = %d\n", n, sum);
  } else {
    int sum = (n + 1) * (n + 1) / 4;
    printf("1 + ... + %d = %d\n", n, sum);
  }
  return 0;
}
```

Câu lệnh if 1 nhánh

Câu lệnh if 2 nhánh.

Các câu lệnh if lồng nhau

Câu lệnh if *có thể có câu lệnh thành phần là câu lệnh* if *và cứ tiếp tục như vậy tạo thành cấu trúc lựa chọn phức tạp với nhiều tầng điều kiện.*

```
 7  #include <stdio.h>
 8
 9  int main() {
10    printf("Nhập 2 số nguyên a và b: ");
11    int a, b;
12    scanf("%d%d", &a, &b);
13    if (a >= 0)
14      if (b < 0) printf("Khác dấu\n");
15      else printf("Cùng dấu\n");
16    else if (b <0) printf("Cùng dấu\n");
17         else printf("Khác dấu\n");
18    return 0;
19  }
```

```
bangoc:$gcc -o prog vd5-3a.c
bangoc:$./prog
Nhập 2 số nguyên a và b: 3 5
Cùng dấu
bangoc:$./prog
Nhập 2 số nguyên a và b: 3 -1
Khác dấu
bangoc:$./prog
Nhập 2 số nguyên a và b: -1 0
Khác dấu
bangoc:$./prog
Nhập 2 số nguyên a và b: -1 -2
Cùng dấu
```

Vấn đề: *Có thể khó xác định nhánh else nào gắn với nhánh if nào.*

Giải pháp: *Kết hợp chia khối và chỉ sử dụng các cấu trúc if … else tuần tự, và có thể tìm cách diễn đạt giải thuật theo cách khác.*

Ví dụ 4.11a. Gỡ rối các lệnh if lồng nhau

Chia khối và chỉ sử dụng các lệnh if … else … tuần tự,

và có thể tìm cách diễn đạt giải thuật theo cách khác

```c
#include <stdio.h>

int main() {
  printf("Nhập 2 số nguyên a và b: ");
  int a, b;
  scanf("%d%d", &a, &b);
  if (a >= 0) {
    if (b < 0) {
      printf("Khác dấu\n");
    } else {
      printf("Cùng dấu\n");
    }
  } else if (b < 0) {
    printf("Cùng dấu\n");
  } else {
    printf("Khác dấu\n");
  }
  return 0;
}
```

```c
#include <stdio.h>

int main() {
  printf("Nhập 2 số nguyên a và b: ");
  int a, b;
  scanf("%d%d", &a, &b);
  if (a >= 0 && b >= 0) {
    printf("Cùng dấu\n");
  } else if (a < 0 && b < 0) {
    printf("Cùng dấu\n");
  } else {
    printf("Khác dấu\n");
  }
  return 0;
}
```

Ví dụ 4.11b. Lựa chọn theo mức

Chương trình này quy đổi điểm thang 10 sang điểm chữ

```c
#include <stdio.h>
int main() {
  float score;
  printf("Nhập điểm = ");
  scanf("%f", &score);
  if (score > 10) {
    printf("Không hợp lệ\n");
  } else if (score >= 9.5 ) {
    printf("A+\n");
  } else if (score >= 8.5) {
    printf("A\n");
  } else if (score >= 8.0) {
    printf("B+\n");
  }
  /* ... */
  return 0;
}
```

Thử hoàn thành chương trình với các thang điểm còn lại

Câu lệnh switch

- Định dạng:

```
switch (biểu-thức) {
    case giá-trị-1: các câu lệnh
    case giá-trị-2: các câu lệnh
    …
    default: câu lệnh
}
```

- Điều kiện:
  - Biểu thức điều khiển phải có kiểu số nguyên.
  - Mỗi nhãn case phải có 1 giá trị số nguyên duy nhất sau khi nâng kiểu.
  - Có thể có tối đa 1 nhãn default.

Câu lệnh switch(2)

- Ý nghĩa:
  - Câu lệnh **switch** chuyển *điều khiển lệnh* tới nhãn **case** có giá trị nhãn bằng giá trị biểu thức điều khiển và bỏ qua phần mã nguồn đứng trước nhãn case đó.
  - Nếu không tồn tại nhãn **case** nào có giá trị nhãn bằng giá trị biểu thức điều khiển thì con trỏ lệnh được chuyển tới nhãn **default** (nếu có), *hoặc kết thúc thực hiện lệnh* **switch** *nếu không có nhãn* **default**.

Ví dụ 4.12a. Rẽ nhánh với switch

```c
#include <stdio.h>

int main() {
  int n;
  printf("Nhập 1 số nguyên trong khoảng [0,..,9]: ");
  scanf("%d", &n);
  switch (n) {
    case 0: printf("Không\n");
    case 1: printf("Một\n");
    case 2: printf("Hai\n");
    case 3: printf("Ba\n");
    case 4: printf("Bốn\n");
    case 5: printf("Năm\n");
    case 6: printf("Sáu\n");
    case 7: printf("Bẩy\n");
    case 8: printf("Tám\n");
    case 9: printf("Chín\n");
    default: printf("Ngoài khoảng\n");
  }
  return 0;
}
```

Được bỏ qua khi nhập 5

```
Nhập 1 số nguyên trong khoảng [0,..,9]: 5
Năm
Sáu
Bẩy
Tám
Chín
Ngoài khoảng
bangoc:$./prog
Nhập 1 số nguyên trong khoảng [0,..,9]: 10
Ngoài khoảng
```

Ví dụ 4.12b. Khai báo biến trong lệnh switch

Phân tích các vấn đề với biến s trong ví dụ sau:

```
 6 #include <stdio.h>
 7
 8 int main() {
 9   int n;
10   printf("Nhập n = ");
11   scanf("%d", &n);
12   switch (n) {
13     int s = 0;
14     case 1: s += 100;
15     case 2: s += 200;
16     default: printf("s = %d\n", s);
17   }
18   return 0;
19 }
```

Nếu nhập n = 2 thì chương trình xuất kết quả gì ra màn hình?

Thử sửa lỗi liên quan đến biến s

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- **Các câu lệnh**
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

Tổng quan về các câu lệnh lặp

- Các câu lệnh lặp trong C:

while (*biểu thức*) *câu lệnh*

do *câu lệnh* **while** (*biểu thức*);

for (*biểu thức*$_{opt}$; *biểu thức*$_{opt}$; *biểu thức*$_{opt}$) *câu lệnh*

for (*khai báo*$_{opt}$; *biểu thức*$_{opt}$; *biểu thức*$_{opt}$) *câu lệnh*

- Câu lệnh lặp khiến câu lệnh thành phần / thân vòng lặp được thực hiện nhiều lần cho tới khi biểu thức điều khiển có giá trị == 0.
- Câu lệnh lặp là khối với phạm vi là tập con của phạm vi chứa nó. Thân vòng lặp cũng là **khối** với **phạm vi** là tập con của phạm vi của cả câu lệnh lặp.

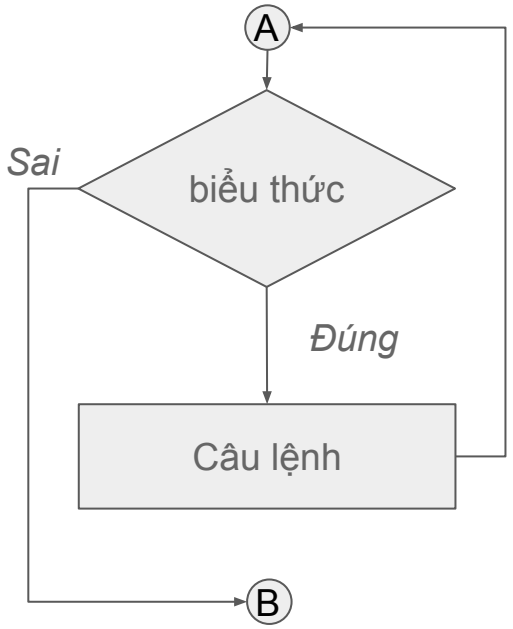
Câu lệnh while

- Định dạng:
  - **while** (*biểu thức*) *câu lệnh*
- Điều kiện:
  - Biểu thức điều khiển thỏa mãn các điều kiện để thực hiện phép so sánh !=, (biểu thức) != 0;
- Ý nghĩa:
  - Khi thực hiện câu lệnh **while** nếu biểu thức điều khiển có giá trị != 0 thì câu lệnh / thân vòng lặp được thực hiện rồi sau đó điều khiển lệnh quay lại vị trí kiểm tra biểu thức,
  - nếu ngược lại (biểu thức điều khiển == 0) thì kết thúc thực hiện câu lệnh **while**.

Với câu lệnh **while** *nếu biểu thức điều khiển có giá trị == 0 từ khi bắt đầu thì thân vòng lặp không được thực hiện lần nào.*

Lưu đồ thực hiện câu lệnh while

while (*biểu thức*) *câu lệnh*



Ví dụ 4.13. Minh họa câu lệnh while

vd5-5.c

```
 6  #include <stdio.h>
 7
 8  int main() {
 9    int n;
10    printf("Nhập số n: ");
11    scanf("%d", &n);
12    while (n > 0) {
13      printf("%5d%*c\n", n, n, '*');
14      --n;
15    }
16    return 0;
17  }
```

Biểu thức điều khiển

Thân vòng lặp

Câu lệnh while

```
bangoc:$./prog
Nhập số n: 8
    8       *
    7      *
    6     *
    5    *
    4   *
    3  *
    2 *
    1*
bangoc:$./prog
Nhập số n: -2
bangoc:$
```

- Với n = 8 thân vòng lặp được thực hiện 8 lần tương ứng với các giá trị n = 8, 7, …, 1.
- Với n = -2 thân vòng lặp không được thực hiện lần nào (được bỏ qua).

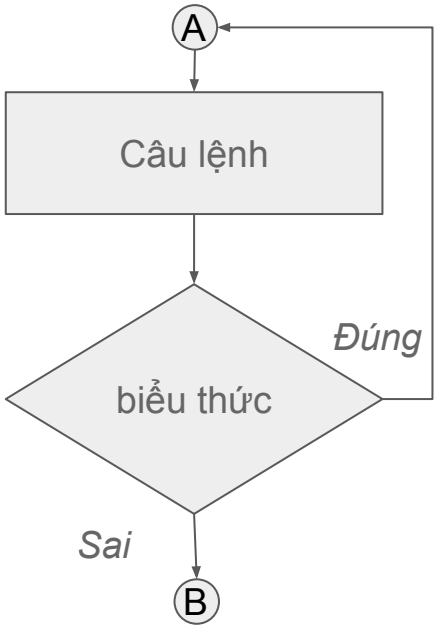
Câu lệnh do while

- Định dạng:
  - **do** *câu lệnh* **while** (*biểu thức*);
- Điều kiện:
  - Biểu thức điều khiển thỏa mãn các điều kiện để thực hiện phép so sánh !=, (biểu thức) != 0.
- Ý nghĩa:
  - Khi thực hiện câu lệnh **do while** trước tiên câu lệnh / thân vòng lặp được thực hiện (lần đầu).
  - Sau đó nếu biểu thức điều khiển có giá trị != 0 thì thực hiện câu lệnh / thân vòng lặp rồi tiếp tục kiểm tra biểu thức điều khiển, nếu ngược lại (biểu thức điều khiển == 0) thì kết thúc câu lệnh do while.

Với câu lệnh **do while** *thân vòng lặp luôn được thực hiện ít nhất 1 lần.*

Lưu đồ thực hiện câu lệnh do while

do *câu lệnh* **while** (*biểu thức*);



Ví dụ 4.14. Minh họa câu lệnh do while

```
7  #include <stdio.h>
8  int main() {
9    int n;
10   do {
11     printf("Nhập số nguyên n > 0: ");
12     scanf("%d", &n);
13   } while (n <= 0);
14   printf("n = %d\n", n);
15   return 0;
16 }
```

*Câu lệnh **do while***

Biểu thức điều khiển

Thân vòng lặp

Chương trình yêu cầu người dùng nhập 1 số nguyên > 0 và yêu cầu nhập lại nếu điều kiện > 0 không được đáp ứng.

Thử: Kết hợp với Ví dụ 4.13

```
bangoc:$gcc -o prog vd4-14.c
bangoc:$./prog
Nhập số nguyên n > 10: 3
Nhập số nguyên n > 10: 10
Nhập số nguyên n > 10: 15
n = 15
bangoc:$
```

Câu lệnh for

- Định dạng:
  - **for** (*mô tả 1*$_{opt}$*; biểu thức 2*$_{opt}$*; biểu thức 3*$_{opt}$)
    câu lệnh
  - Mô tả 1 có thể là khai báo (C99), thường được sử dụng để khai báo và khởi tạo biến điều khiển lặp.
    - Biến được khai báo trong mô tả 1 có thể được sử dụng trong phạm vi vòng lặp for: trong biểu thức 2, biểu thức 3 và thân vòng lặp.
  - Mô tả 1 cũng có thể là biểu thức, thường được sử dụng để khởi tạo các biến điều khiển lặp.
  - Biểu thức 2 là biểu thức điều khiển, được kiểm tra trước khi thực hiện câu lệnh / thân vòng lặp. *Nếu được để trống thì sẽ được tự động thay thế bằng 1 giá trị != 0 (luôn đúng).*
  - Biểu thức 3 thường được sử dụng để thay đổi các biến điều khiển lặp.
  - Mô tả 1 và biểu thức 3 là các thành phần không bắt buộc.

Câu lệnh for(2)

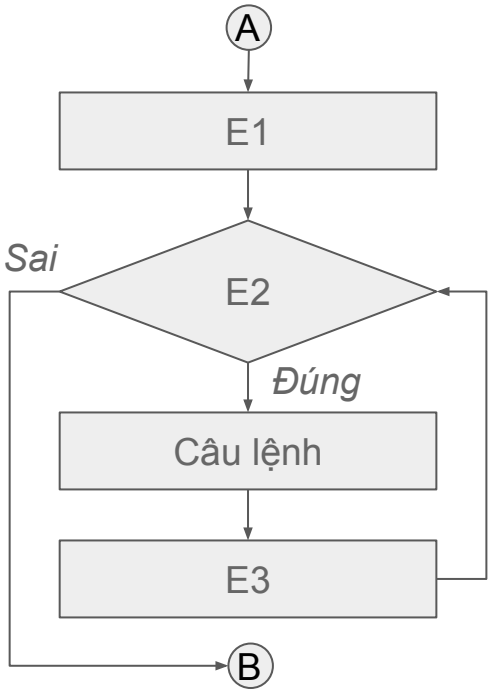
- Điều kiện:
  - Các biến được khai báo trong mô tả 1 nếu có phải thuộc phân lớp lưu trữ auto hoặc register.
  - Biểu thức 2 phải thỏa mãn các điều kiện để thực hiện phép so sánh !=, (biểu thức 2) != 0.
- Ý nghĩa:
  - Khi thực hiện câu lệnh **for** trước tiên mô tả 1 được thực hiện, sau đó nếu biểu thức 2 có giá trị != 0 thì thực hiện thân vòng lặp, rồi thực hiện biểu thức 3 (nếu có), sau đó con trỏ lệnh được di chuyển quay lại vị trí kiểm tra biểu thức 2, nếu ngược lại (biểu thức 2 có giá trị == 0) thì kết thúc vòng lặp.

Mô tả 1 (nếu có) luôn được thực hiện 1 lần, và nếu biểu thức 2 == 0 ngay sau đó thì thân vòng lặp và biểu thức 3 không được thực hiện lần nào.

Lưu đồ thực hiện câu lệnh for

Trường hợp đầy đủ 3 biểu thức:

```
for (E1; E2; E3) câu lệnh
```



Ví dụ 4.15. Minh họa vòng lặp for

vd5-7.c

```
6  #include <stdio.h>
7
8  int main() {
9    int n;
10   printf("Nhập n = ");
11   scanf("%d", &n);
12   for (int i = 1; i <= n; ++i) {
13     printf("%5d%*c\n", i, i, '*');
14   }
15   return 0;
16 }
```

Khai báo 1 (C99)

Biểu thức 2

Biểu thức 3

Câu lệnh for

Thân vòng lặp

```
bangoc:$./prog
Nhập n = 8
    1*
    2 *
    3  *
    4   *
    5    *
    6     *
    7      *
    8       *
bangoc:$./prog
Nhập n = -3
bangoc:$
```

- Với n = 8 thân vòng lặp được thực hiện 8 lần với các giá trị i = 1, 2, 3, …, 8.
- Với n = -3 thân vòng lặp không được thực hiện lần nào (được bỏ qua).

Ví dụ 4.15b. Minh họa vòng lặp for(2)

Các mã nguồn tương đương.

```c
#include <stdio.h>

int main() {
  int n;
  printf("Nhập n = ");
  scanf("%d", &n);
  int i = 1;
  for (; i <= n;++i) {
    printf("%5d%*c\n", i, i, '*');
  }
  return 0;
}
```

*Câu lệnh **for** này không có mô tả 1.*

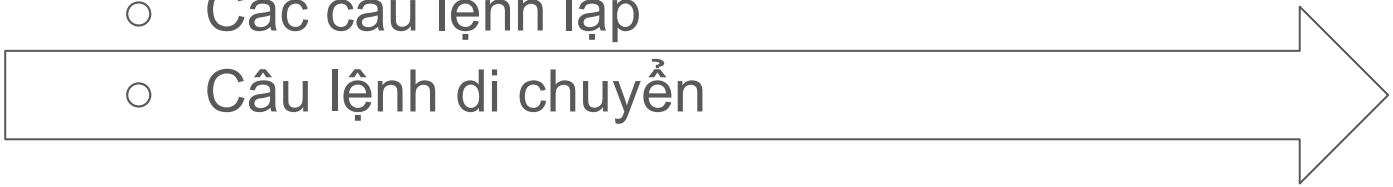
```c
#include <stdio.h>

int main() {
  int n;
  printf("Nhập n = ");
  scanf("%d", &n);
  int i;
  for (i = 1; i <= n;) {
    printf("%5d%*c\n", i, i, '*');
    ++i;
  }
  return 0;
}
```

*Trong câu lệnh **for** này mô tả 1 là biểu thức khởi tạo; không có biểu thức 3.*

*Câu lệnh **for** rất linh hoạt, và có thể được sử dụng theo nhiều cách khác nhau.*

Nội dung

- Nhập và xuất theo định dạng
  - Xuất theo định dạng
  - Nhập theo định dạng
- Biểu thức lô-gic
- **Các câu lệnh**
  - Các khái niệm
  - Câu lệnh tính biểu thức và câu lệnh null
  - Câu lệnh được gán nhãn
  - Câu lệnh gộp
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển



Các câu lệnh di chuyển

Câu lệnh di chuyển khiến điều khiển lệnh chuyển tức thời đến 1 vị trí khác, (có thể) không theo thứ tự tuần tự:

- goto định-danh; : Di chuyển đến 1 câu lệnh được gán nhãn, xu hướng hiện đại hạn chế sử dụng goto *(mã nguồn chứa goto có thể khó hiểu và khó quản lý)*.
- return /*... */; : Kết thúc thực hiện hàm và trả về giá trị (chi tiết sẽ được cung cấp sau cùng với hàm)
- break ; : Kết thúc thực hiện lệnh switch hoặc lệnh lặp trực tiếp chứa nó.
- continue; : Bắt đầu vòng lặp mới tức thời, giống như vừa kết thúc thực hiện thân vòng lặp.

Nội dung chi tiết về break và continue được cung cấp trong bài giảng này

Câu lệnh continue

- Định dạng:
  - continue;
- Điều kiện:
  - Chỉ được sử dụng trong vòng lặp.
- Ý nghĩa:
  - Khi được thực hiện lệnh **continue** khiến điều khiển lệnh di chuyển tới điểm kết thúc của thân vòng lặp trực tiếp chứa nó, thực hiện lệnh được tiếp diễn giống như sau khi kết thúc thực hiện thân vòng lặp.

Tuy lệnh **goto** *có thể thay thế* **continue**, *nhưng xu hướng hiện đại hạn chế sử dụng* **goto.**

continue và goto trong vòng lặp while

```
while (/*...*/)
{
  /*...*/
  continue;
  /*...*/
}
```

Sau đó

Kiểm tra biểu thức điều khiển, ...

Tương đương

```
while (/*...*/)
{
  /*...*/
  goto contin;
  /*...*/
contin: ;
}
```

continue và goto trong vòng lặp do while

```
do {
  /*...*/
  continue;
  /*...*/
} while (/*...*/);
```

Tương đương

```
do {
  /*...*/
  goto contin;
  /*...*/
contin: ;
} while (/*...*/);
```

Sau đó

Kiểm tra biểu thức điều khiển, ...

continue và goto trong vòng lặp for

```
for (/*...*/) {
  /*...*/
  continue;
  /*...*/
}
```

Tương đương

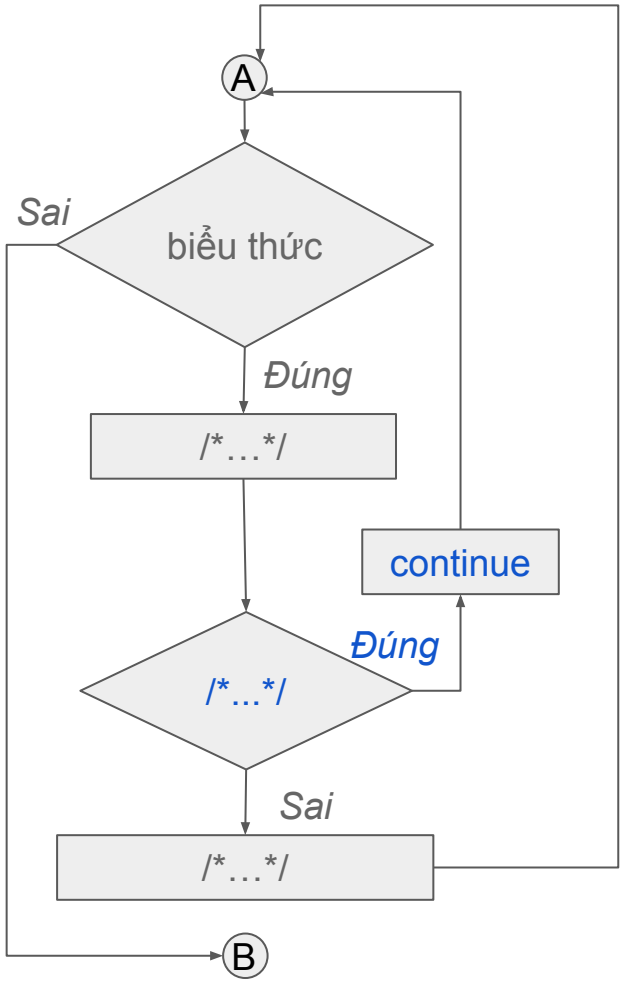
```
for (/*...*/) {
  /*...*/
  goto contin;
  /*...*/
contin: ;
}
```

Sau đó

Thực hiện biểu thức 3 (nếu có),
Kiểm tra biểu thức 2, ...

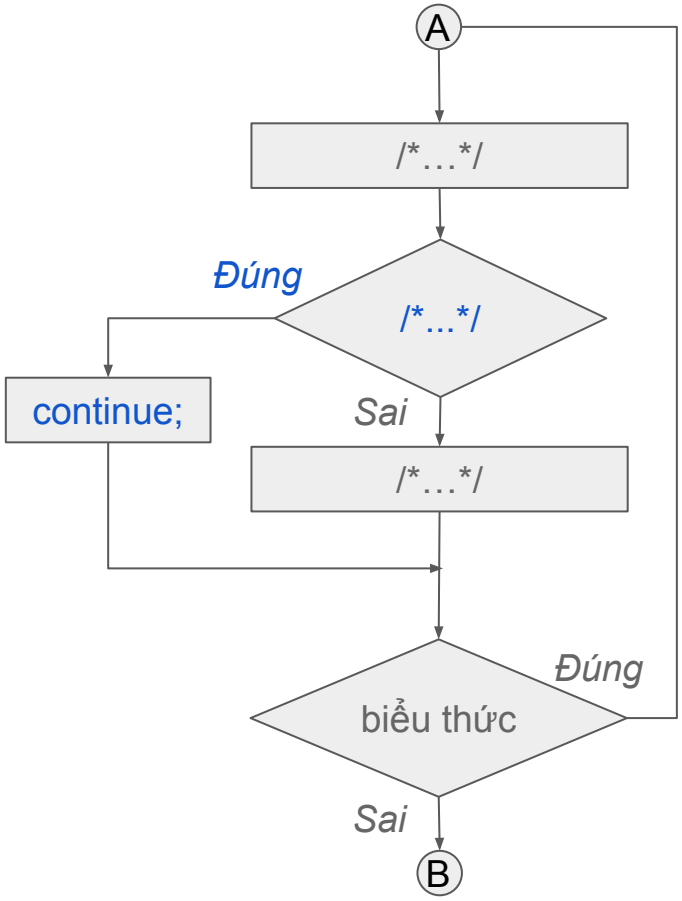
Lưu đồ thực hiện vòng lặp while có continue

```
while (biểu thức) {
   /*...*/
   if (/*...*/) continue;
   /*...*/
}
```



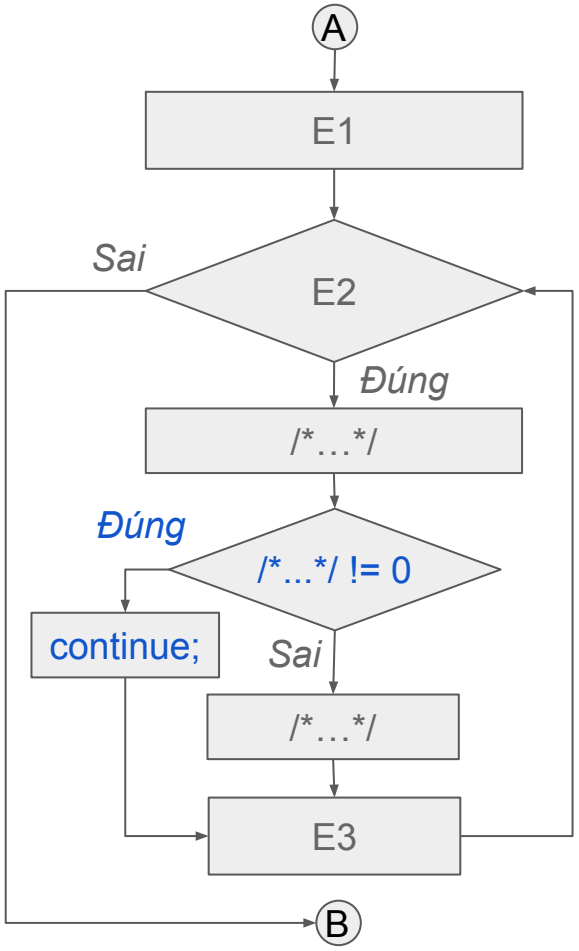
Lưu đồ vòng lặp do while có continue

```
do {
    /* … */
    if (/*...*/) continue;
    /* … */
} while ( biểu thức );
```



Lưu đồ thực hiện vòng lặp for có continue

```
for (E1; E2; E3) {
    /* … */
    if (/*...*/) continue;
    /* … */
}
```



Câu lệnh break

- Định dạng:
  - break;
- Điều kiện:
  - Câu lệnh **break** chỉ được sử dụng trong phạm vi câu lệnh **switch** hoặc câu lệnh lặp.
- Ý nghĩa:
  - Khi được thực hiện lệnh **break** làm kết thúc tức thời câu lệnh **switch** hoặc câu lệnh lặp *trực tiếp chứa nó*.

Tuy câu lệnh **goto** *có thể thay thế* **break**, *nhưng xu hướng hiện đại hạn chế sử dụng* **goto.**

break và goto trong vòng lặp while

```
while (/*...*/)
{
  /*...*/
  break;
  /*...*/
}
```

Sau đó

Giống như sau khi kết thúc vòng lặp

Tương đương

```
while (/*...*/)
{
  /*...*/
  goto brk;
  /*...*/
}
brk: ;
```

break và goto trong vòng lặp do while

```
do {
  /*...*/
  break;
  /*...*/
} while (/*...*/);
```

Tương đương

```
do {
  /*...*/
  goto brk;
  /*...*/
} while (/*...*/);
brk: ;
```

Sau đó

Giống như sau khi kết thúc vòng lặp

break và goto trong vòng lặp for

```
for (/*...*/) {
  /*...*/
  break;
  /*...*/
}
```

Tương đương

```
for (/*...*/) {
  /*...*/
  goto brk;
  /*...*/
}
brk: ;
```

Sau đó

Giống như sau khi kết thúc vòng lặp

break và goto trong vòng câu lệnh switch

```
switch (biểu thức)
  /*...*/
  break;
  /*...*/
}
```

Tương đương

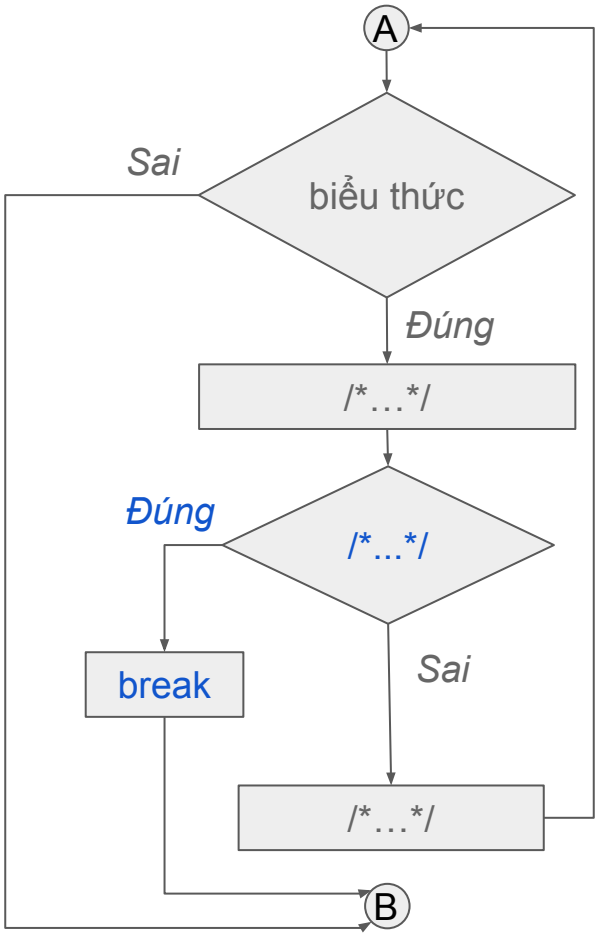
```
switch (/*...*/) {
  /*...*/
  goto brk;
  /*...*/
}
brk: ;
```

Sau đó

Giống như sau khi kết thúc lệnh switch

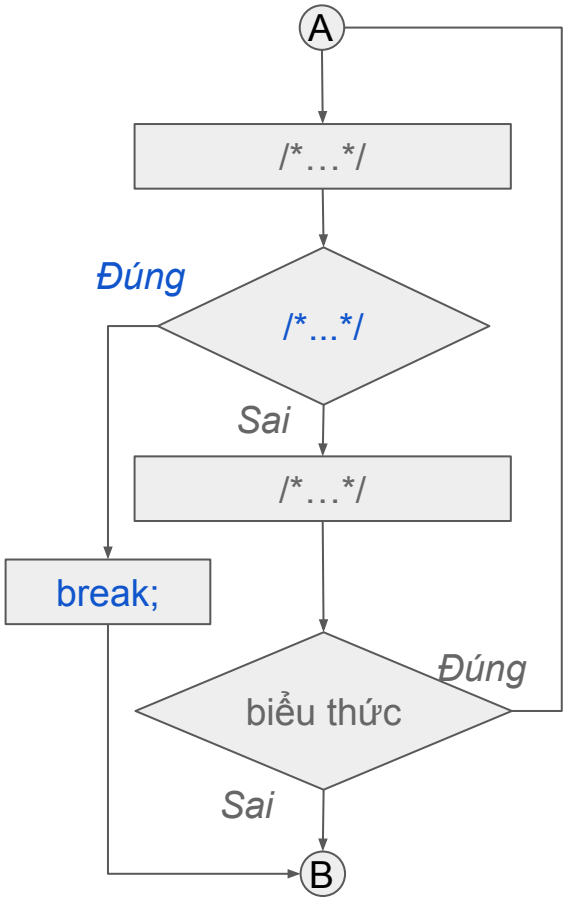
Lưu đồ thực hiện vòng lặp while có break

```
while (biểu thức) {
    /* … */
    if (/*...*/) break;
    /* … */
}
```



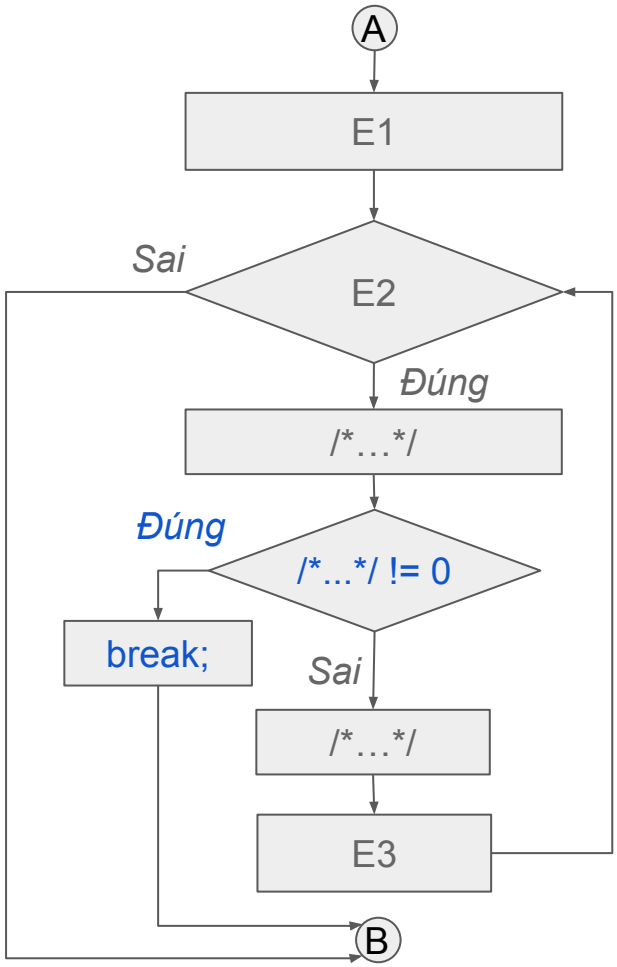
Lưu đồ vòng lặp do while có break

```
do {
     /* … */
     if (/*...*/) break;
     /* … */
} while ( biểu thức );
```



Lưu đồ thực hiện vòng lặp for có break

```
for (E1; E2; E3) {
    /* … */
    if (/*...*/) break;
    /* … */
}
```



Ví dụ 4.16. Sử dụng break trong switch

```
bangoc:$gcc -o prog vd5-8.c
bangoc:$./prog
Nhập 1 số nguyên trong khoảng [0,..,9]: 5
Năm
bangoc:$./prog
Nhập 1 số nguyên trong khoảng [0,..,9]: 10
Ngoài khoảng
bangoc:$
```

vd5-8.c

```c
#include <stdio.h>

int main() {
  int n;
  printf("Nhập 1 số nguyên trong khoảng [0,..,9]: ");
  scanf("%d", &n);
  switch (n) {
    case 0: printf("Không\n"); break;
    case 1: printf("Một\n"); break;
    case 2: printf("Hai\n"); break;
    case 3: printf("Ba\n"); break;
    case 4: printf("Bốn\n"); break;
    case 5: printf("Năm\n"); break;
    case 6: printf("Sáu\n"); break;
    case 7: printf("Bẩy\n"); break;
    case 8: printf("Tám\n"); break;
    case 9: printf("Chín\n"); break;
    default: printf("Ngoài khoảng\n");
  }
  return 0;
}
```

Được bỏ qua khi nhập 5

Kết thúc câu lệnh switch

Vòng lặp vô hạn và break

- Vòng lặp vô hạn là vòng lặp có biểu thức điều khiển luôn luôn != 0.
- Chúng ta có thể lựa chọn kết thúc vòng lặp ở vị trí bất kỳ trong thân vòng lặp bằng câu lệnh break.
  - Nếu thỏa mãn điều kiện dừng (đk dừng) thì break;

```
for (;;) {
  /* ... */
  if ( đk dừng ) break;
  /* ... */
}
```

```
while (1) {
  /* ... */
  if ( đk dừng ) break;
  /* ... */
}
```

```
do {
  /* ... */
  if ( đk dừng ) break;
  /* ... */
} while (1);
```

Có thể lựa chọn vị trí bất kỳ để kiểm tra điều kiện dừng