

# Ngôn ngữ lập trình C

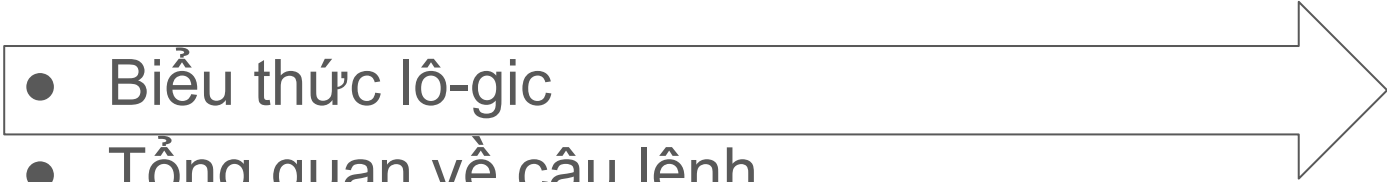
## Bài 5. Biểu thức lô-gic và các câu lệnh

*Soạn bởi: TS. Nguyễn Bá Ngọc*

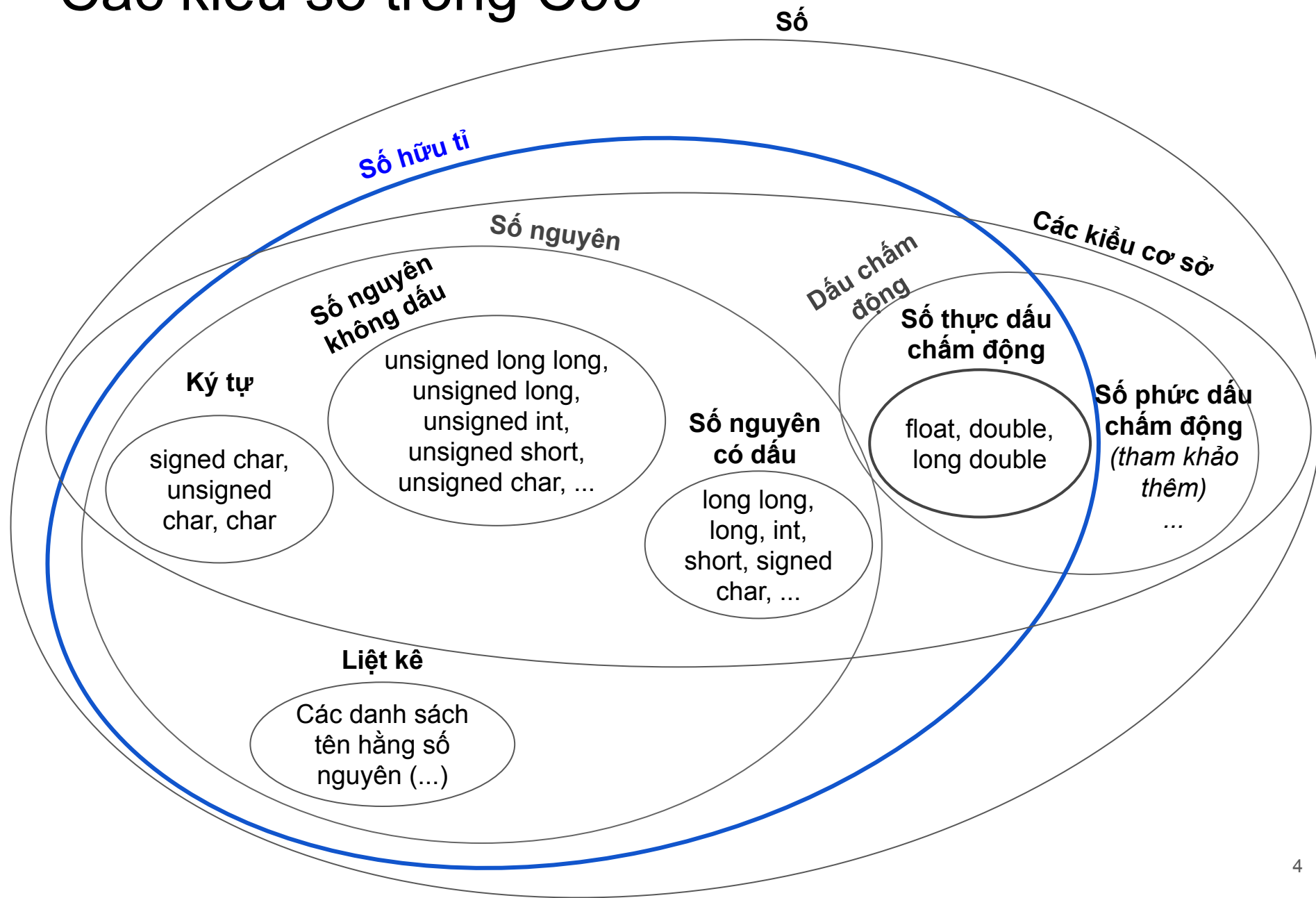
# Nội dung

- Biểu thức lô-gic
- Tổng quan về câu lệnh
- Các câu lệnh rẽ nhánh
- Các câu lệnh lặp
- Câu lệnh di chuyển

# Nội dung

- 
- Biểu thức lô-gic
  - Tổng quan về câu lệnh
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển

# Các kiểu số trong C99



# Các giá trị chân lý

- Các giá trị chân lý trong lô-gic được biểu diễn bằng các macro trong `stdbool.h` (C99)
  - `bool` được chuyển thành `_Bool`
  - `true` - 1
  - `false` - 0
- Kết quả ép kiểu 1 giá trị đơn (scalar value) thành giá trị kiểu `_Bool` là 0 nếu giá trị ban đầu `== 0`, là 1 nếu ngược lại - giá trị ban đầu `!= 0`.

# Các toán tử thứ tự

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

# Các toán tử so sánh

- 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:
  - Quy tắc cân bằng kiểu số được áp dụng cho các toán hạng.
  - Kết quả có kiểu int và  $= 1$  nếu đúng,  $= 0$  nếu ngược lại.
- Ví dụ:
  - $2 == 3$  cho kết quả  $= 0$
  - $2 != 3$  cho kết quả  $= 1$
- Với 2 toán hạng bất kỳ,  $a != b \Leftrightarrow !(a == b)$ , trong 2 biểu thức  $a == b$  và  $a != b$  chỉ có duy nhất 1 biểu thức 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:
  - Kết quả có kiểu int và  $= 1$  nếu cả 2 toán hạng  $\neq 0$ ,  $= 0$  nếu ngược lại - có ít nhất 1 toán hạng  $= 0$ .
  - **Các toán hạng** được thực hiện từ trái sang phải.
  - Toán hạng ở vế trái 1 đơn vị tuần tự
  - Nếu vế trái có giá trị  $= 0$  thì kết quả biểu thức sai và không được tính vế phải.



# Toán tử lô-gic OR

- Định dạng:
  - $a \parallel b$
- Điều kiện:
  - Các toán hạng có kiểu số.
  - *(Trường hợp toán hạng có kiểu con trỏ sẽ được học sau)*
- Ý nghĩa:
  - Kết quả có kiểu int và  $= 1$  nếu có ít nhất 1 trong 2 toán hạng có giá trị  $\neq 0$ ,  $= 0$  nếu ngược lại - cả 2 toán hạng đều  $= 0$ .
  - Các toán hạng được thực hiện theo thứ tự từ trái sang phải.
  - Toán hạng ở vế trái là 1 đơn vị tuần tự
  - Nếu vế trái có giá trị  $\neq 0$  thì biểu thức đúng và không được tính vế phải.

## Ví dụ 5.1.a. Biểu thức lô-gic AND

```
int a = 1;
```

```
int b = 1;
```

**(a % 2 == 0 && ++b > 1)** cho kết quả = 0, sau đó b = 1 (ngắt sớm)

```
a = 2;
```

**(a % 2 == 0 && ++b > 1)** cho kết quả = 1, sau đó b = 2 (++b được thực hiện).

## Ví dụ 5.1.b. Biểu thức lô-gic OR

`int a = 2, b = 1;`

`(a % 2 == 0 || ++b % 2 == 0)` Cho kết quả = 1, sau đó `b = 1` (ngắt sớm).

`a = 3;`

`(a % 2 == 0 || ++b % 2 == 0)` Cho kết quả = 1, `b = 2` (`++b` được thực hiện).

# Toán tử phủ định

- Định dạng:
  - !a
- Điều kiện:
  - Toán hạng có kiểu số.
  - *(Trường hợp toán hạng có kiểu con trỏ sẽ được học sau)*
- Ý nghĩa:
  - Kết quả có kiểu int và = 1 nếu toán hạng có giá trị == 0, = 0 nếu ngược lại - toán hạng != 0.
  - Tương đương với  $0 == a$ .
- Ví dụ:  
int a = 0, b = 1;  
!b Cho kết quả == 0  
!(a == b) Cho kết quả == 1 (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*).
- 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 cân bằng kiểu số cho b và c.
- Ví dụ:
  - $2 < 3 ? 10 : 3.5$  cho kết quả = 10, kiểu double.

## Ví dụ 5.2. Cân bằng kiểu số trong so sánh

```
int x = -1;
```

```
unsigned y = 10;
```

Biểu thức nào cho kết quả đúng:

$x < y$

$x > y$

*Đổi chiều kết quả với chương trình C?*

## Ví dụ 5.3. So sánh số thực

```
double x = 0.1, y = 10.1 - 10;
```

Biểu thức so sánh nào cho kết quả đúng?


$x < y$

$x > y$

$x == y$

*Đổi chiều kết quả với chương trình C?*

# Nội dung

- Biểu thức lô-gic
  - Tổng quan về câu lệnh
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển
- 



# Định nghĩa và phân loại câu lệnh

- Câu lệnh đặc tả 1 hành động được thực hiện. Các lệnh trong 1 dãy lệnh được thực hiện tuần tự nếu không có chỉ dẫn cụ thể khác.
  - Ví dụ: break, continue
- 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

- Các mô tả (khai báo, câu lệnh) có thể được gom lại thành 1 đơn vị cú pháp được gọi là 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.
- 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.*

# 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.*

# 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** (*các chi tiết sẽ được cung cấp sau*).
- 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; // (*Xu hướng hiện đại hạn chế sử dụng goto bởi vì khó nắm bắt luồng di chuyển và dễ tạo ra lỗi*)

# Câu lệnh gộp

- Có thể gom nhiều khai báo và câu lệnh trong 1 cặp dấu {}, đoạn 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 {}
  - Câu lệnh gộp cũng có thể là thành phần của 1 câu lệnh gộp khác lớn hơn, 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 5.4. Câu lệnh gộp và phạm vi

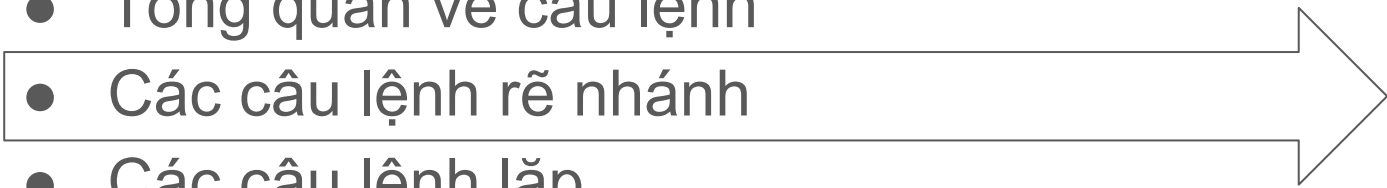
```
1.  #include <stdio.h>
2.  int main()                               S1
3.  {
4.      int s = 0;                             S2
5.      {
6.          int s = 100;                       S3
7.          {
8.              int s = 202;
9.              printf("s = %d\n", s);
10.         }
11.         printf("s = %d\n", s);
12.     }
13.     printf("s = %d\n", s);
14. }
```

*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 khác nhau.*

# Nội dung

- Biểu thức lô-gic
  - Tổng quan về câu lệnh
  - Các câu lệnh rẽ nhánh
  - Các câu lệnh lặp
  - Câu lệnh di chuyển
- 

# Các câu lệnh rẽ nhánh

- Câu lệnh rẽ nhánh lựa chọn 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.



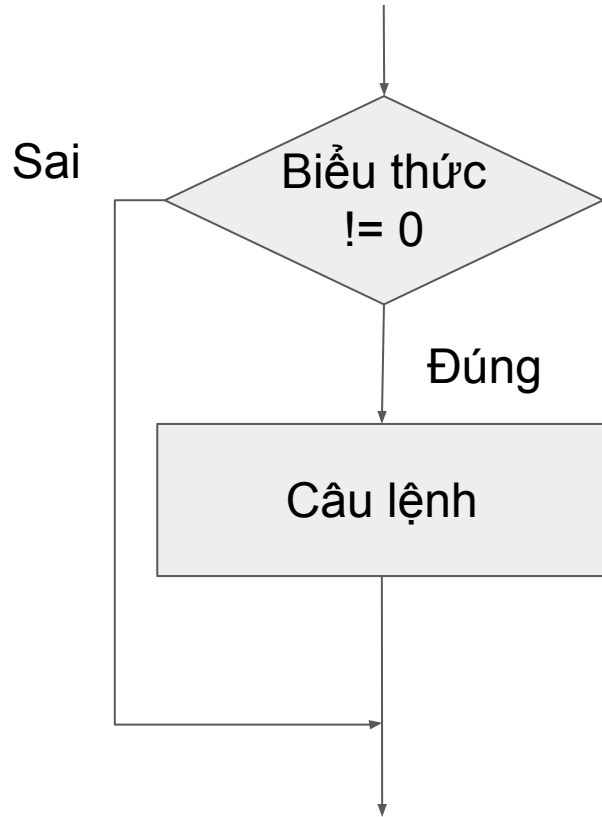
# Các câu lệnh if

# Câu lệnh if

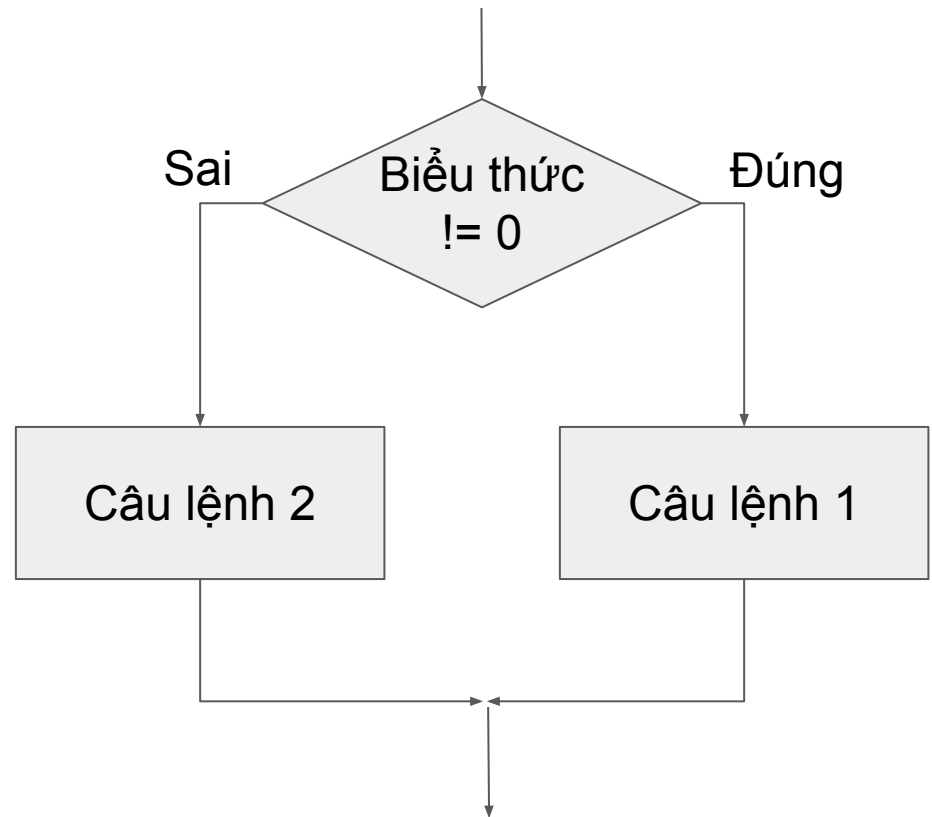
- Định dạng
  - 1 nhánh: **if** (*biểu thức*) *câu lệnh*
  - 2 nhánh: **if** (*biểu thức*) *câu lệnh 1* **else** *câu lệnh 2*
- Điều kiện
  - Biểu thức điều khiển thỏa mãn các điều kiện cần thiết cho phép so sánh  $\neq$ , (*biểu thức*)  $\neq 0$
- Ý nghĩa:
  - **if** (*biểu thức*) *câu lệnh*: Câu lệnh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

# Mẫu lưu đồ thực hiện lệnh **if**

**if** (biểu thức) câu-lệnh



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



# Ví dụ 5.5. Câu lệnh if

Nhập 1 số nguyên dương: -1  
-1 không thỏa mãn yêu cầu.

Nhập 1 số nguyên dương: 5  
 $1 + \dots + 5 = 9$

Nhập 1 số nguyên dương: 6  
 $2 + \dots + 6 = 12$

```
1.  #include <stdio.h>
2.  int main() {
3.      int n;
4.      printf("Nhập 1 số nguyên dương: ");
5.      scanf("%d", &n);
6.      if (n <= 0) {
7.          printf("%d không thỏa mãn yêu cầu.\n", n);
8.          return 0;
9.      }
10.     if (n % 2 == 0) {
11.         int sum = n * (n + 2) / 4;
12.         printf("2 + ... + %d = %d\n", n, sum);
13.     } else {
14.         int sum = (n + 1) * (n + 1) / 4;
15.         printf("1 + ... + %d = %d\n", n, sum);
16.     }
17. }
```

*Câu lệnh if 1 nhánh*

*Câu lệnh if 2 nhánh.*

# Các câu lệnh if lồng nhau

- Câu lệnh if có thể là câu lệnh thành phần của 1 câu lệnh if khác và có thể hình thành cấu trúc lựa chọn phức tạp với nhiều tầng điều kiện.
- **Vấn đề:** *Khó đọc mã nguồn - 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 5.6a. Các lệnh if lồng nhau

*Chương trình so sánh dấu của 2 số nguyên*

```
1.  #include <stdio.h>   Hãy ghép nối nhánh else với  
2.  int main() {        nhánh if của nó? Khó xác định?  
3.    printf("Nhập 2 số nguyên a và b: ");  
4.    int a, b;  
5.    scanf("%d%d", &a, &b);  
6.    if (a >= 0)  
7.        if (b < 0) printf("Khác dấu\n");  
8.        else printf("Cùng dấu\n");  
9.    else if (b < 0) printf("Cùng dấu\n");  
10.   else printf("Khác dấu\n");  
11. }
```

Nhập 2 số nguyên a và b: 1 3  
Cùng dấu

Nhập 2 số nguyên a và b: -1 2  
Khác dấu

## Ví dụ 5.6b. Các lệnh if lồng nhau<sub>(2)</sub>

*Có thể sử dụng câu lệnh gộp và các lệnh if ... else ... tuần tự,*

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

Hãy ghép nối nhánh **else** với nhánh **if** của nó? Khó xác định?

## Ví dụ 5.6c. Các lệnh if lồng nhau<sub>(3)</sub>

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

*Hãy ghép nối nhánh **else** với nhánh **if** của nó? Khó xác định?*

```
1. #include <stdio.h>
2. int main() {
3.     printf("Nhập 2 số nguyên a và b: ");
4.     int a, b;
5.     scanf("%d%d", &a, &b);
6.     if (a >= 0 && b >= 0) {
7.         printf("Cùng dấu\n");
8.     } else if (a < 0 && b < 0) {
9.         printf("Cùng dấu\n");
10.    } else {
11.        printf("Khác dấu\n");
12.    }
13. }
```



## Ví dụ 5.7. Lựa chọn theo mức

*Chương trình quy đổi điểm thang 10 sang điểm chữ*

```
1.  #include <stdio.h>
2.  int main() {
3.      float score;
4.      printf("Nhập điểm = ");
5.      scanf("%f", &score);
6.      if (score > 10) {
7.          printf("Không hợp lệ\n");
8.      } else if (score >= 9.5) {
9.          printf("A+\n");
10.     } else if (score >= 8.5) {
11.         printf("A\n");
12.     } else if (score >= 8.0) {
13.         printf("B+\n");
14.     }
15. }
```

*Hãy hoàn thành chương trình với các thang điểm còn lại?*

# Câu lệnh **switch**

# Câu lệnh switch

## Định dạng phổ biến

*Biểu thức điều khiển, phải có kiểu số nguyên*

**switch** (biểu-thức)

{

**case** hằng-biểu-thức-1: dãy-lệnh

**case** hằng-biểu-thức-2: dãy-lệnh

...

**default:** dãy-lệnh

}

*Phải có giá trị duy nhất sau khi ép kiểu*

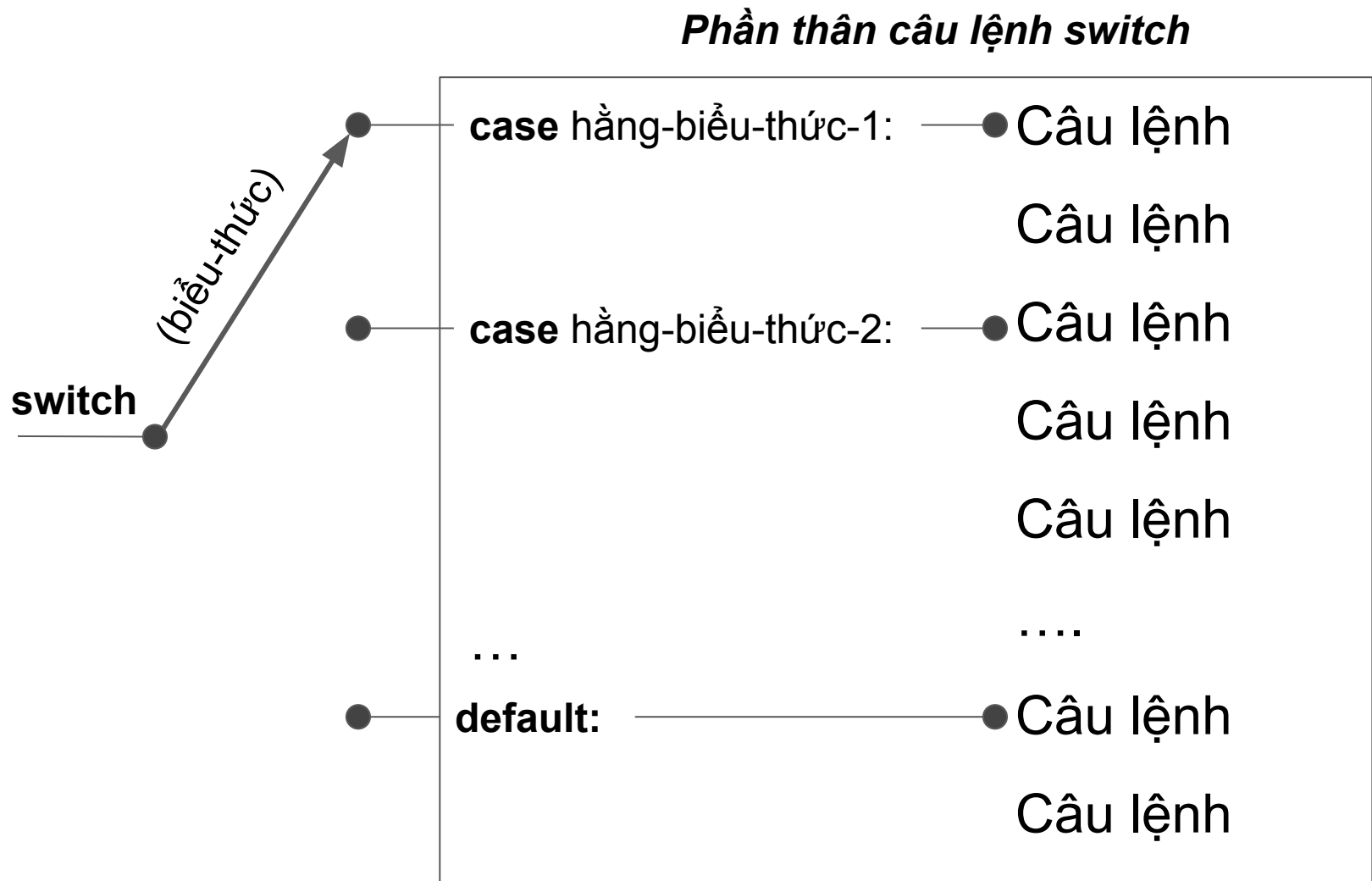
*Được có tối đa 1 nhãn default trực tiếp trong 1 lệnh **switch** (lệnh **switch** nằm bên trong có thể có nhãn **default** riêng).*

*Phần thân của câu lệnh **switch**.*

# Ý nghĩa của câu lệnh **switch**

- Biểu thức điều khiển được nâng kiểu số nguyên và giá trị của các nhãn **case** được ép kiểu thành kiểu sau khi nâng của biểu thức điều khiển.
- Nếu giá trị sau khi ép kiểu của biểu thức điều khiển khớp với giá trị sau khi ép kiểu của 1 nhãn **case** thì điều khiển lệnh được chuyển tới vị trí sau nhãn **case** đó. Trong trường hợp ngược lại, nếu có 1 nhãn **default** trực tiếp thì điều khiển lệnh được chuyển tới sau vị trí nhãn **default** đó.
  - Trong trường hợp di chuyển tới 1 nhãn thì toàn bộ phần thân lệnh **switch** đứng trước nhãn đó được bỏ qua.
- Nếu không có nhãn **case** nào khớp với biểu thức điều khiển và không có nhãn **default** trực tiếp nào thì lệnh **switch** kết thúc mà không có phần nào trong thân lệnh **switch** được thực hiện.

# Mô hình lệnh switch theo định dạng phổ biến



Từ switch trong tiếng Anh có nghĩa là cái chuyển mạch

# Ví dụ 5.8a. Rẽ nhánh với switch

```
1. #include <stdio.h>
2. int main() {
3.     int n;
4.     printf("Nhập 1 số nguyên trong khoảng [0,...,9]: ");
5.     scanf("%d", &n);
6.     switch (n) {
7.         case 0: printf("Không\n");
8.         case 1: printf("Một\n");
9.         case 2: printf("Hai\n");
10.        case 3: printf("Ba\n");
11.        case 4: printf("Bốn\n");
12.        case 5: printf("Năm\n");
13.        case 6: printf("Sáu\n");
14.        case 7: printf("Bảy\n");
15.        case 8: printf("Tám\n");
16.        case 9: printf("Chín\n");
17.        default: printf("Ngoài khoảng\n");
18.    }
19. }
```

*Được bỏ qua khi nhập 5*

Nhập 1 số nguyên trong  
khoảng [0,...,9]: 5  
Năm  
Sáu  
Bảy  
Tám  
Chín  
Ngoài khoảng

Có thể sử dụng **break** để kết thúc lệnh **switch**

# Ví dụ 5.8b. Rẽ nhánh với switch<sub>(2)</sub>

```
1.  #include <stdio.h>
2.  int main() {
3.      int n;
4.      printf("Nhập 1 số nguyên trong khoảng [0,...,9]: ");
5.      scanf("%d", &n);
6.      switch (n) {
7.          case 0: printf("Không\n"); break;
8.          case 1: printf("Một\n"); break;
9.          case 2: printf("Hai\n"); break;
10.         case 3: printf("Ba\n"); break;
11.         case 4: printf("Bốn\n"); break;
12.         case 5: printf("Năm\n"); break;
13.         case 6: printf("Sáu\n"); break;
14.         case 7: printf("Bảy\n"); break;
15.         case 8: printf("Tám\n"); break;
16.         case 9: printf("Chín\n"); break;
17.         default: printf("Ngoài khoảng\n");
18.     }
19. }
```

Nhập 1 số nguyên trong khoảng [0,...,9]: 5  
Năm  
Nhập 1 số nguyên trong khoảng [0,...,9]: 10  
Ngoài khoảng

*Được bỏ qua khi nhập 5*

*Kết thúc câu lệnh switch*

*Chi tiết về lệnh **switch** sẽ được cung cấp sau trong bài này*

# Ví dụ 5.9. Vấn đề khai báo biến trong **switch**

Phân tích các vấn đề với biến *s*

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

```
int main() {
```

```
    int n;
```

```
    printf("Nhập n = ");
```

```
    scanf("%d", &n);
```

```
    switch (n) {
```

```
        int s = 100;
```

```
        case 1: s += 100;
```

```
        case 2: s += 200;
```

```
        default: printf("s = %d\n", s);
```

```
    }
```

```
}
```

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

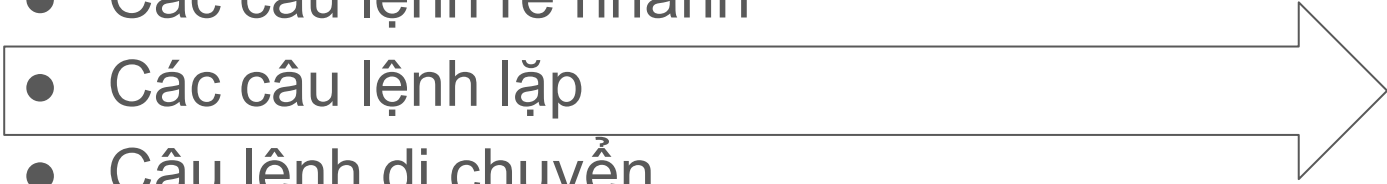
Nhập n = 2 s = 200
-----------------------

*Thử sửa lỗi liên quan đến biến *s**

*Gợi ý: Đưa biến *s* ra bên ngoài, phía trước lệnh **switch***



# Nội dung

- Biểu thức lô-gic
  - Tổng quan về câu lệnh
  - 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 thứcopt; biểu thứcopt ; biểu thứcopt) câu Lệnh`

`for ( khai báoopt ; biểu thứcopt ; biểu thứcopt) 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**

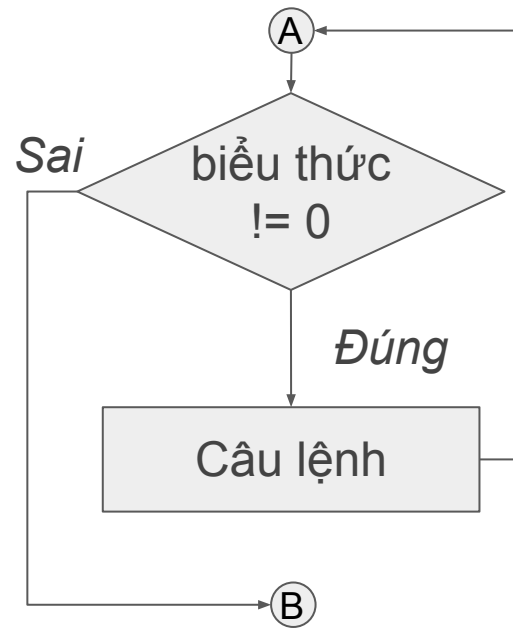
# Câu lệnh while

- 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)  $\neq 0$ ;
- Ý nghĩa:
  - Khi thực hiện câu lệnh **while** nếu biểu thức điều khiển có giá trị  $\neq 0$  thì câu lệnh / thân vòng lặp được thực hiện rồi sau đó điều khiển lệnh quay lại kiểm tra biểu thức điều khiển,
  - 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.*

# Mẫu lưu đồ thực hiện câu lệnh while

**while** ( *biểu thức* ) *câu Lệnh*



# Ví dụ 5.10. Minh họa câu lệnh while

```
1. #include <stdio.h>
2. int main() {
3.     int n;
4.     printf("Nhập số n: ");
5.     scanf("%d", &n);
6.     while (n > 0)
7.     {
8.         printf("%5d%*c\n", n, n, '*');
9.         --n;
10.    }
11. }
```

*Biểu thức điều khiển* (điểm đến dòng 6)

*Thân vòng lặp* (điểm đến dòng 7-9)

*Câu lệnh while* (điểm đến dòng 6)

Nhập số n: 8

```
      8      *
      7      *
      6      *
      5      *
      4      *
      3      *
      2      *
      1      *
```

Nhập số n: -2

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

# Câu lệnh do

# Câu lệnh **do**

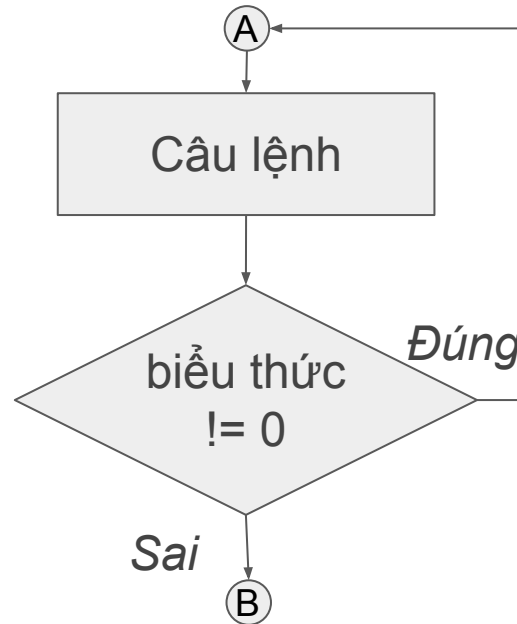
- Định dạng:
  - *do câu lệnh while ( biểu thức );*
- Điều kiện:
  - Biểu thức điều khiển thỏa mãn các điều kiện để thực hiện phép so sánh (biểu thức)  $\neq 0$ .
- Ý nghĩa:
  - Khi thực hiện câu lệnh **do** trước tiên câu lệnh / thân vòng lặp được thực hiện (lần đầu).
  - Sau đó nếu biểu thức điều khiển có giá trị  $\neq 0$  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**.

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



# Mẫu lưu đồ thực hiện câu lệnh **do**

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



## Ví dụ 5.11. Minh họa câu lệnh **do**

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 5.10 ?*

```
1.  #include <stdio.h>
2.  int main() {
3.      int n;
4.      do {
5.          printf("Nhập n > 0 = ");
6.          scanf("%d", &n);
7.      } while (n <= 0);
8.  }
```

Nhập n > 0 = -1

Nhập n > 0 = -2

Nhập n > 0 = 10

Câu lệnh **for**

# 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  $\neq 0$  (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**<sub>(2)</sub>

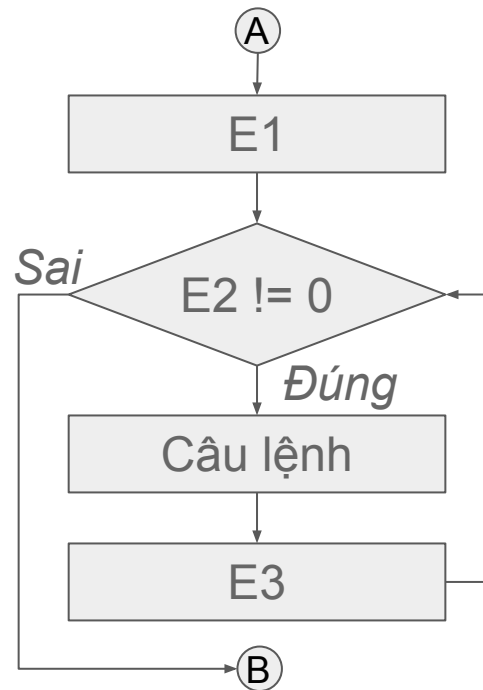
- Điều kiện:
  - Các biến được khai báo trong mô tả 1 nếu có phải thuộc phân lớp lưu trữ **auto** hoặc **register**.
  - Biểu thức 2 phải thỏa mãn các điều kiện để thực hiện phép so sánh (biểu thức 2)  $\neq 0$ .
- Ý nghĩa:
  - Khi thực hiện câu lệnh **for** trước tiên mô tả 1 được thực hiện, sau đó nếu biểu thức 2 có giá trị  $\neq 0$  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.*

# Mẫu lưu đồ thực hiện câu lệnh **for**

*Trường hợp đầy đủ 3 biểu thức:*

for (E1; E2; E3) câu lệnh



## Ví dụ 5.12a. Minh họa vòng lặp **for**

```
1. #include <stdio.h>
2. int main() {
3.     int n;
4.     printf("Nhập n = ");
5.     scanf("%d", &n);
6.     for (int i = 1; i <= n; ++i) {
7.         printf("%5d%*c\n", i, i, '*');
8.     }
9. }
```

*Khai báo 1 (C99)*

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

```
Nhập n = 8
  1*
 2 *
3  *
4   *
5    *
6     *
7      *
8       *
```

## Ví dụ 5.12b. Minh họa vòng lặp **for**<sub>(2)</sub>

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

```
1.  #include <stdio.h>
2.  int main() {
3.      int n;
4.      printf("Nhập n = ");
5.      scanf("%d", &n);
6.      int i = 1;
7.      for (; i <= n; ++i) {
8.          printf("%5d%c", i, i, '*');
9.      }
10. }
```

*Không có mô tả 1.*



## Ví dụ 5.12c. Minh họa vòng lặp **for**<sub>(3)</sub>

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

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

*Mô tả 1 là biểu thức khởi tạo; không có biểu thức 3.*

# Toán tử liệt kê

- Định dạng:
  - Biểu thức 1, biểu thức 2 *(trong đó dấu phẩy “,” là toán tử liệt kê)*
- Điều kiện:
  - Khác ngữ cảnh mà dấu , được sử dụng để phân tách các phần tử của 1 danh sách (ví dụ biểu thức gọi hàm).
- Ý nghĩa:
  - Toán hạng ở vế trái được thực hiện nhưng kết quả được bỏ qua (giống như được ép kiểu thành void), và có 1 điểm tuần tự sau khi thực hiện.
  - Sau đó toán hạng ở vế phải được thực hiện, kết quả của nó đồng thời là kết quả của biểu thức.
- Ví dụ:
  - `int x, y;`
  - `x = 3, y = 5, x == y` - Có giá trị là `0` (false), kiểu `int`.

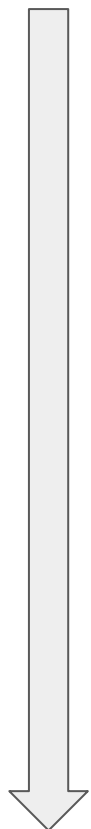
# Gộp nhiều biểu thức

*Biểu thức liệt kê về hình thức giống như biểu thức gộp của nhiều biểu thức thành phần*

- Ý nghĩa của biểu thức trong ví dụ  $x = 3, y = 5, x == y$ :
  - Đầu tiên thực hiện biểu thức  $x = 3$ , sau đó thực hiện biểu thức  $y = 5$ , và cuối cùng thực hiện biểu thức  $x == y$ .
  - Kết quả của biểu thức  $x == y$  (biểu thức cuối cùng) đồng thời là kết quả của biểu thức liệt kê.
- Thường được sử dụng để thực hiện nhiều biểu thức thành phần trong ngữ cảnh chỉ cho phép 1 biểu thức
  - Ví dụ có thể khởi tạo nhiều biến trong câu lệnh `for`  
`for (sum = 0, i = 0; i < n; ++i) { ... }`
  - Thực hiện các tính toán trước khi kiểm tra điều kiện lặp  
`while (scanf("%d", &n), n <= 0) { ... }`
  - v.v.

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

Ưu tiên cao  
(thực hiện trước)



Ưu tiên thấp  
(thực hiện sau)

Tên toán tử	Ký hiệu	Ví dụ	Chuỗi
Tăng 1 (hậu tố) Giảm 1 (hậu tố)	++ --	x++ x--	--
Tăng, giảm 1 (tiền tố) Dấu (tiền tố) Đảo dây bit Phủ định Dung lượng	++, -- +, - ~ ! sizeof	++x, --x +x, -x ~x !x sizeof(int)	←
Ép kiểu	()	(double)n	←
Nhân, Chia, Phần dư	*, /, %	x * y, x / y, x % y	⇒
Cộng, trừ	+, -	x + y, x - y	⇒
Dịch trái, dịch phải	<<, >>	x << y, x >> y	⇒
So sánh thứ tự	<, >, <=, >=	x < y	⇒
So sánh bằng	==, !=	x == y	⇒
AND theo bit	&	x & y	⇒
XOR theo bit	^	x ^ y	⇒
OR theo bit		x   y	⇒
AND lô-gic	&&	x && y	⇒
OR lô-gic		x    y	⇒
Lựa chọn	?:	x? y: z	←
Gán đơn giản Gán kết hợp	= *=, /=, %=, +=, -=, <<=, >>=, &=, ^=,  =	x = y x *= y, x /= y, ...	←
Liệt kê	,	s = 0, i = 0	⇒

# Nội dung

- Biểu thức lô-gic
- Tổng quan về câu lệnh
- 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 vì khó nắm bắt luồng di chuyển (khó quản lý mã nguồn) và dễ gây ra lỗi.
- **return** biểu-thức; : Kết thúc thực hiện hàm và trả về giá trị  
*(chi tiết sẽ được cung cấp sau cùng với nội dung về hàm)*
- **break** ; : Kết thúc thực hiện lệnh switch hoặc lệnh lặp trực tiếp chứa nó.
- **continue**; : Bắt đầu vòng lặp mới giống như vừa kết thúc thực hiện thân vòng lặp - Bỏ qua phần thân vòng lặp đứng sau continue.

# Câu lệnh **continue**

# 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 - bỏ qua phần sau lệnh goto trong thân vòng lặp trực tiếp chứa nó, chương trình được tiếp diễn giống như sau khi kết thúc thực hiện thân vòng lặp.



# Mẫu vòng lặp **while** có **continue**

<pre><b>while</b> (biểu thức) {     /*phần trước*/     <b>continue</b>;     /*phần sau*/ }</pre>	<i>Tương đương</i>	<pre><b>while</b> (biểu thức) {     /*phần trước*/     <b>goto</b> contin;     /*phần sau*/ contin: ; }</pre>
--	--------------------	---

*!Các khuôn mẫu mã nguồn chứa goto trong bài giảng này chỉ được đưa ra nhằm mục đích diễn giải ý nghĩa của các lệnh di chuyển khác chứ không được sử dụng trong thực tế (được thay thế bằng các lệnh di chuyển cụ thể hơn).*

# Mẫu vòng lặp do có **continue**

```
do {  
    /*phần trước*/  
    continue;  
    /*phần sau*/  
} while (biểu thức);
```

*Tương đương*

```
do {  
    /*phần trước*/  
    goto contin;  
    /*phần sau*/  
contin: ;  
} while (biểu thức);
```

# Mẫu vòng lặp **for** có **continue**

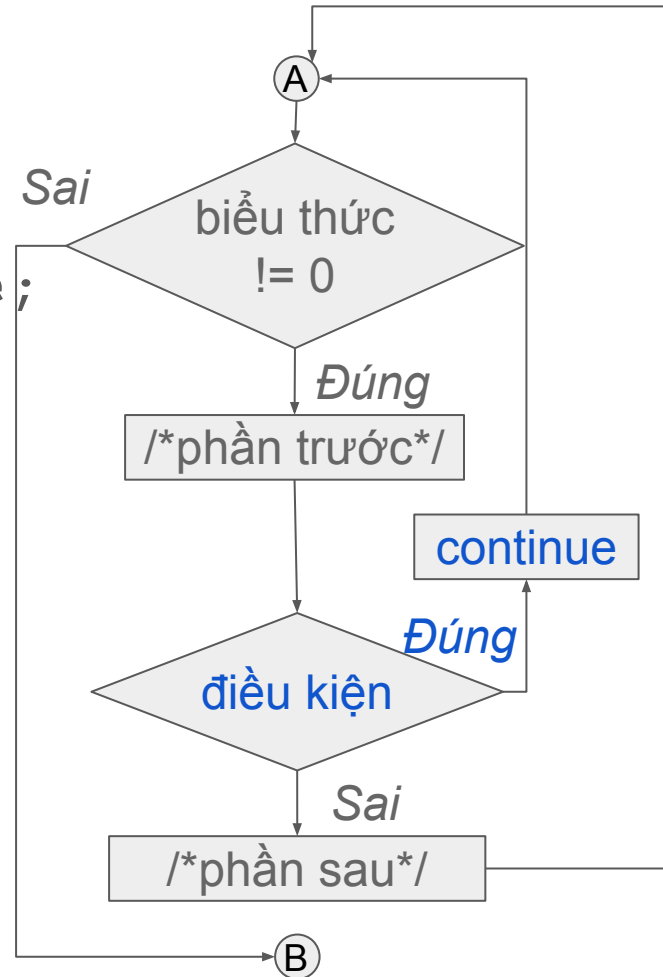
```
for (E1;E2;E3) {  
    /*phần trước*/  
    continue;  
    /*phần sau*/  
}
```

*Tương đương*

```
for (E1;E2;E3) {  
    /*phần trước*/  
    goto contin;  
    /*phần sau*/  
contin: ;  
}
```

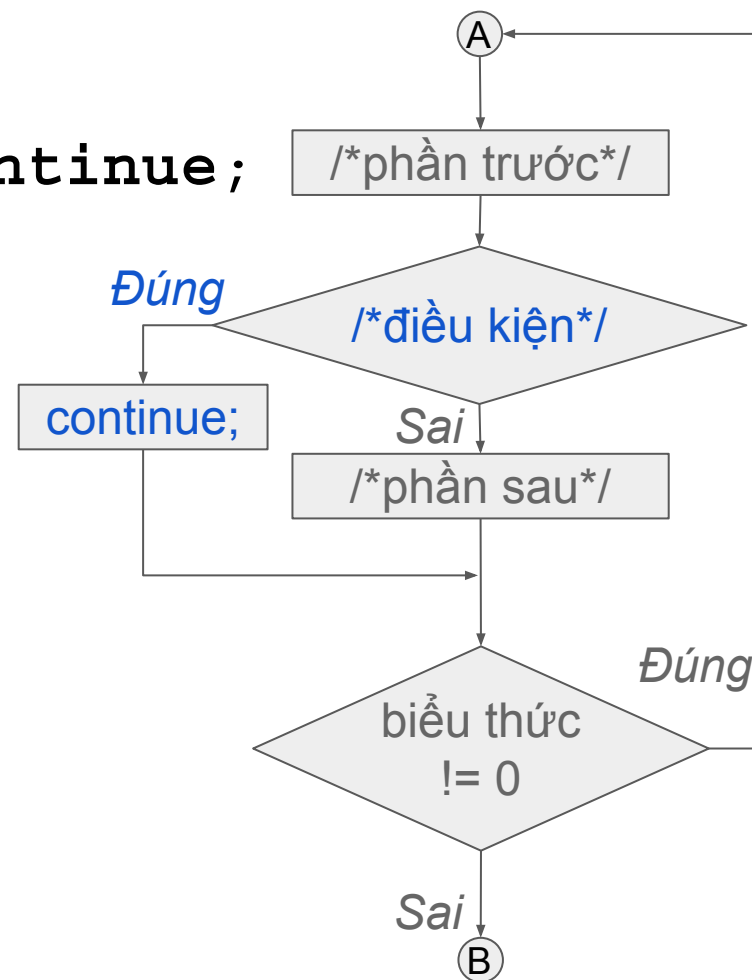
# Mẫu lưu đồ vòng lặp **while** có **continue**

```
while (biểu thức) {  
    /*phần trước*/  
    if (điều kiện) continue;  
    /*phần sau*/  
}
```



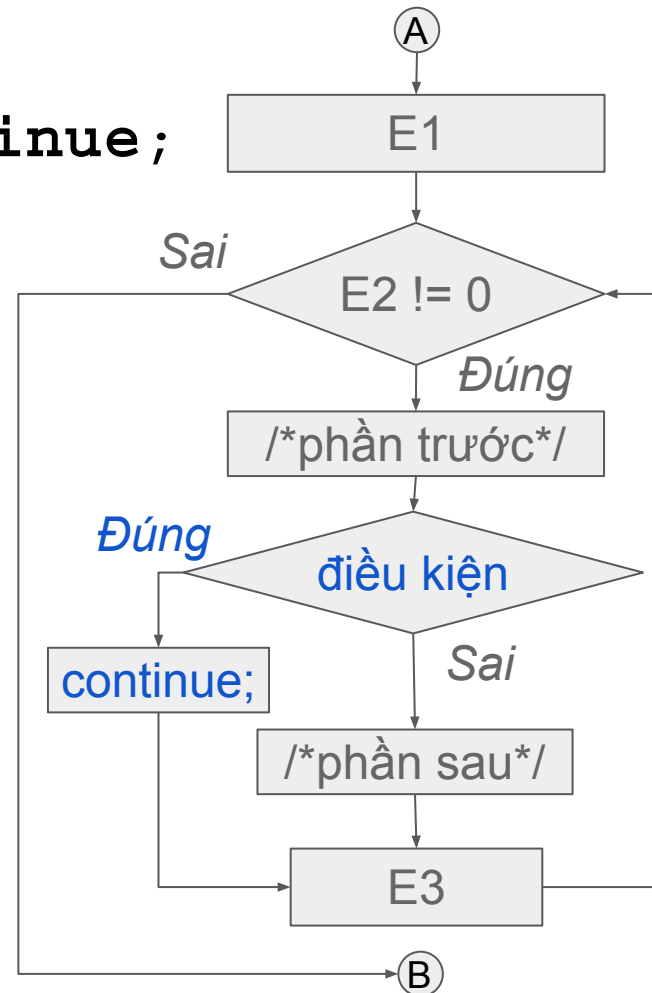
# Mẫu lưu đồ vòng lặp do có **continue**

```
do {  
    /*phần trước*/  
    if (/*điều kiện*/) continue;  
    /*phần sau*/  
} while ( biểu thức );
```



# Mẫu lưu đồ vòng lặp **for** có **continue**

```
for (E1; E2; E3) {  
    /*phần trước*/  
    if (/*điều kiện*/) continue;  
    /*phần sau*/  
}
```



Câu lệnh **break**

# 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*.



# Mẫu vòng lặp **while** có **break**

```
while (biểu thức) {  
    /*phần trước*/  
    break;           Tương đương  
    /*phần sau*/  
}
```

```
while (biểu thức) {  
    /*phần trước*/  
    goto brk;  
    /*phần sau*/  
}  
brk: ;
```

# Mẫu vòng lặp **do** có **break**

```
do {  
    /*phần trước*/  
    break;  
    /*phần sau*/  
} while (biểu thức);
```

*Tương đương*

```
do {  
    /*phần trước*/  
    goto brk;  
    /*phần sau*/  
} while (biểu thức);  
brk: ;
```

# Mẫu vòng lặp **for** có **break**

```
for (E1;E2;E3) {  
    /*phần trước*/  
    break;  
    /*phần sau*/  
}
```

*Tương đương*

```
for (E1;E2;E3) {  
    /*phần trước*/  
    goto brk;  
    /*phần sau*/  
}  
brk: ;
```

# Mẫu câu lệnh **switch** có **break**

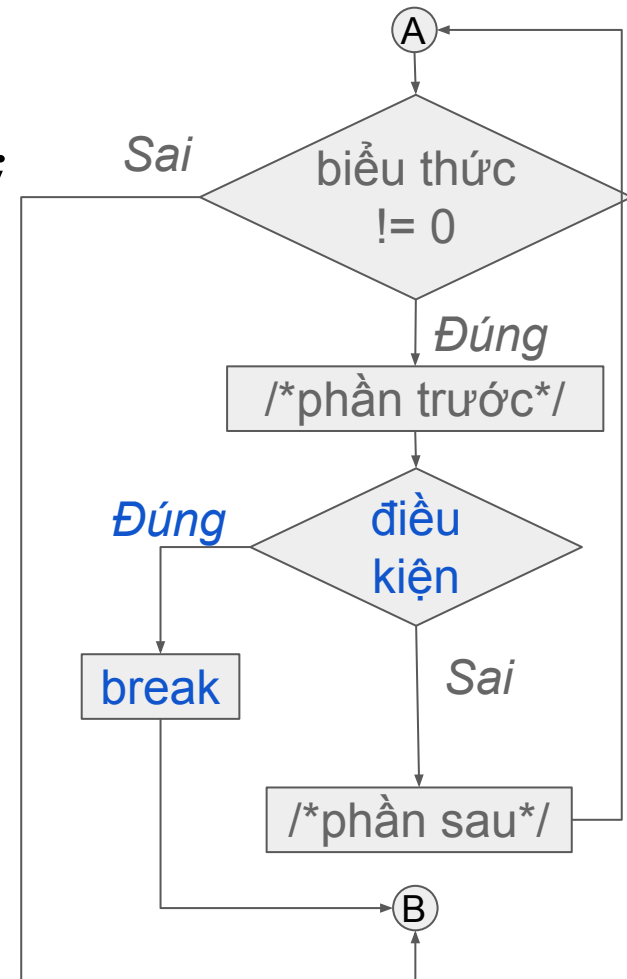
```
switch (biểu thức)
    /*phần trước*/
break;
    /*phần sau*/
}
```

*Tương đương*

```
switch (biểu thức) {
    /*phần trước*/
    goto brk;
    /*phần sau*/
}
brk: ;
```

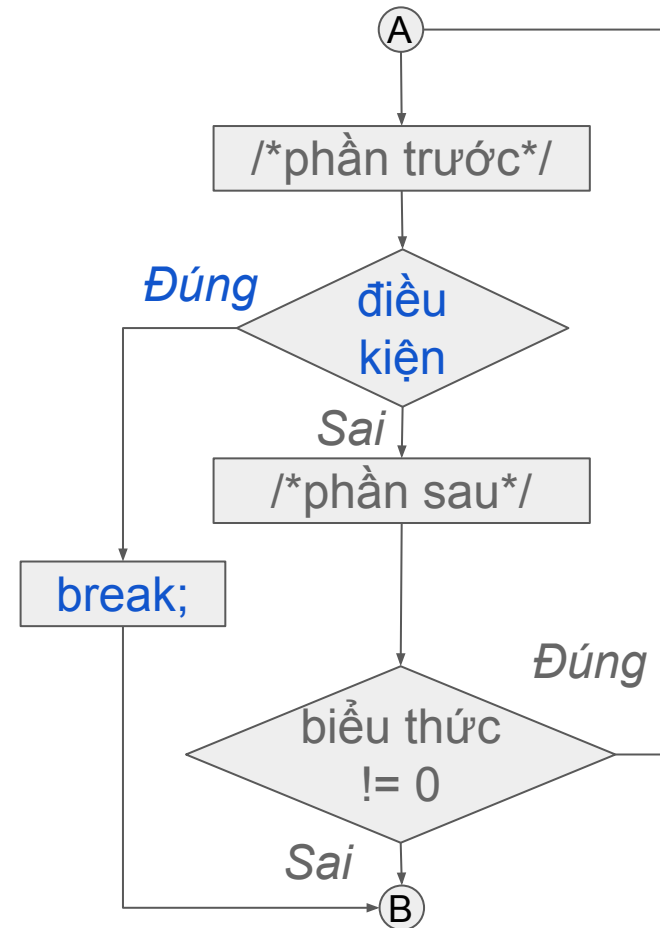
# Mẫu lưu đồ vòng lặp **while** có **break**

```
while (biểu thức) {  
    /*phần trước*/  
    if (/*điều kiện*/) break;  
    /*phần sau*/  
}
```



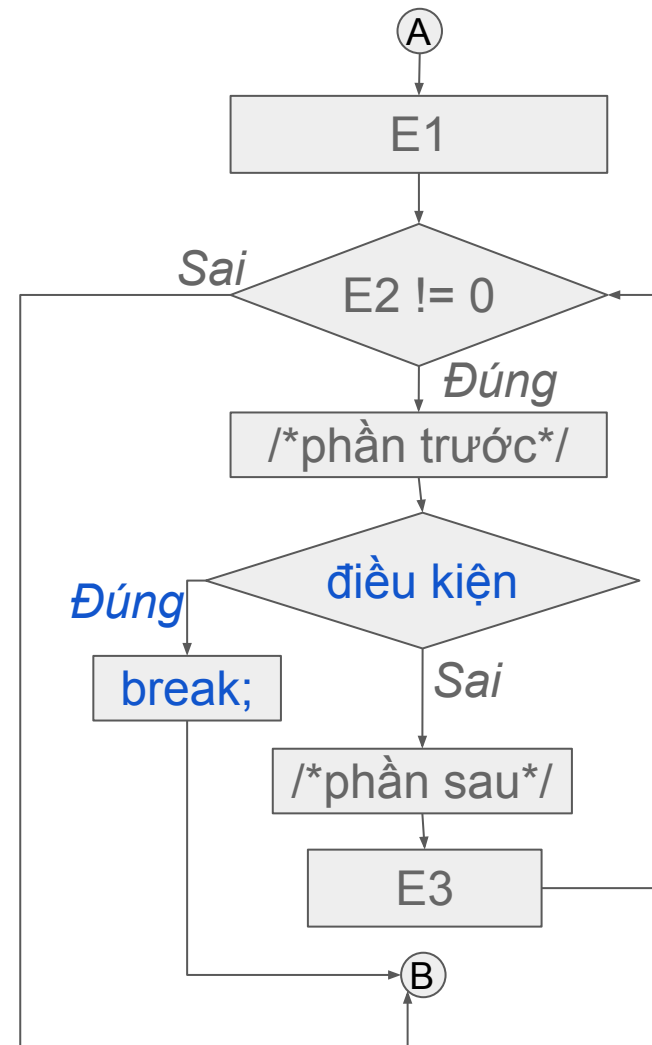
# Mẫu lưu đồ vòng lặp **do** có **break**

```
do {  
    /*phần trước*/  
    if (điều kiện) break;  
    /*phần sau*/  
} while ( biểu thức );
```



# Mẫu lưu đồ vòng lặp **for** có **break**

```
for (E1; E2; E3) {  
    /*phần trước*/  
    if (điều kiện) break;  
    /*phần sau*/  
}
```



# Vòng lặp vô hạn và **break**

- Vòng lặp vô hạn là vòng lặp có biểu thức điều khiển luôn luôn  $\neq 0$ .
- Tuy nhiên 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 vai trò như các lựa chọn khác để diễn đạt lô-gic lặp*



