

# Ngôn ngữ lập trình C

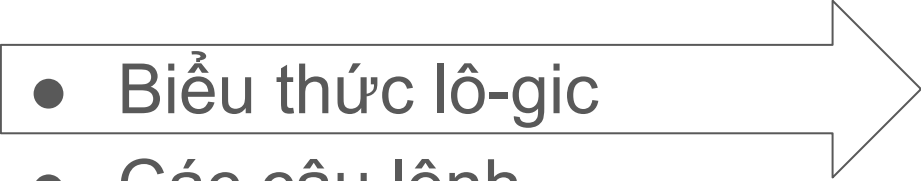
## Bài 5. Biểu thức lô-gic và các câu lệnh

*Soạn bởi: TS. Nguyễn Bá Ngọc*

# Nội dung

- Biểu thức lô-gic
- Các câu lệnh

# Nội dung

- 
- Biểu thức lô-gic
  - Các câu lệnh

# Các toán tử quan hệ so sánh

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

- Quan hệ nhỏ hơn:  $<$ ; lớn hơn:  $>$ ; nhỏ hơn hoặc bằng:  $<=$ ; lớn hơn hoặc bằng:  $>=$
- Biểu thức quan hệ có giá trị bằng 1 nếu quan hệ so sánh được mô tả là đúng, và có giá trị bằng 0 nếu ngược lại. Giá trị biểu thức có kiểu int.
- *Lưu ý*:  $a < b < c$  có thể gặp trong diễn đạt thông thường khác với  $a < b < c$  trong C.
  - Trong C:  $a < b < c$  là một chuỗi liên tiếp các phép so sánh và được tính giống như:  $(a < b) < c$ .
  - Điều kiện  $a < b$  và  $b < c$  được biểu diễn là  $(a < b) \&\& (b < c)$
  - (Tương tự với các toán tử quan hệ khác)

*Chuỗi các toán tử quan hệ được thực hiện theo thứ tự từ trái sang phải.*

# Các toán tử so sánh bằng

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

- Toán tử so sánh bằng: `==`
- Toán tử so sánh khác: `!=`
- Giá trị biểu thức so sánh bằng 1 nếu quan hệ so sánh là đúng và bằng 0 nếu ngược lại (quan hệ so sánh là sai).  
Giá trị biểu thức có kiểu `int`.
- Với 2 toán hạng bất kỳ, chỉ có duy nhất 1 trong 2 quan hệ so sánh (`==` và `!=`) là đúng.

*Chuỗi các toán tử quan hệ so sánh bằng được thực hiện theo thứ tự từ trái sang phải.*

# Toán tử lô-gic AND

- Ký hiệu: `&&`
- Biểu thức AND có giá trị bằng 1 nếu cả 2 toán hạng đều có giá trị khác 0, nếu ngược lại thì giá trị biểu thức bằng 0. Giá trị biểu thức có kiểu int.
- Toán tử `&&` đảm bảo thứ tự thực hiện các toán hạng 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.

*Chuỗi các toán tử AND được thực hiện theo thứ tự từ trái sang phải nhưng có thể được ngắt sớm khi tính được toán hạng có giá trị `== 0`.*

# Toán tử lô-gic OR

- Ký hiệu: `||`
- Biểu thức OR có giá trị bằng 1 nếu 1 toán hạng bất kỳ trong 2 toán hạng có giá trị khác 0, nếu ngược lại thì giá trị biểu thức bằng 0. Giá trị biểu thức có kiểu int.
- Toán tử `||` giống với toán tử `&&`, cũng đảm bảo thứ tự thực hiện các toán hạng từ trái sang phải.
- Toán hạng thứ nhất là 1 đơn vị tuần tự, nếu có giá trị khác 0 thì toán hạng thứ 2 không được tính.

*Chuỗi toán tử OR được thực hiện theo thứ tự từ trái sang phải nhưng có thể được ngắt sớm khi tính được toán hạng có giá trị  $\neq 0$ .*

# Toán tử lựa chọn

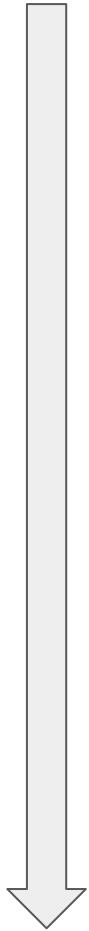
- Ký hiệu: ?:
- Cấu trúc: E1? E2: E3
- Thành phần E1 là 1 đơn vị tuần tự, nếu biểu thức E1 có giá trị khác 0 thì biểu thức E2 được thực hiện (biểu thức E3 không được thực hiện trong trường hợp này) và giá trị biểu thức lựa chọn là giá trị của biểu thức E2, nếu ngược lại ( $E1 == 0$ ) thì biểu thức E3 được thực hiện (biểu thức E2 không được thực hiện trong trường hợp này) và giá trị biểu thức lựa chọn là giá trị của biểu thức E3.
- Ví dụ:
  - $\text{min} = a < b ? a : b;$
  - $\text{grade} \geq 9.0 ? \text{"A+"} : \text{grade} \geq 8.5 ? \text{"A"} : \text{"other"}$

*Biểu thức lựa chọn có thể là toán tử của 1 biểu thức lựa chọn khác*



# 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--)	<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
Quan hệ so sánh	<, >, <=, >=	Trái -> Phải
Quan hệ bằng	==, !=	Trái -> Phải
AND theo bit	& (x & y)	Trái -> Phải
XOR theo bit	^ (x ^ y)	Trái -> Phải
OR theo bit	(x   y)	Trái -> Phải
AND lô-gic	&&	Trái -> Phải
OR lô-gic		Trái -> Phải
Lựa chọn	?:	Phải -> Trái (Rẽ nhánh)
Gán đơn giản Gán kết hợp	= *=, /=, %=, +=, -=, <<=, >>=, &=, ^=,  =	Phải -> Trái

# Nội dung

- Biểu thức lô-gic
  - Các câu lệnh
- 

# Khái niệm khối và các câu lệnh

# Khối

- Khối cho phép gom các khai báo và các câu lệnh thành một đơn vị cú pháp.
  - Đóng gói các khai báo và các câu lệnh trong một phạm vi riêng.
- Các loại khối tiêu biểu trong C gồm có:
  - Câu lệnh gộp
  - Câu lệnh rẽ nhánh
  - Câu lệnh lặp

# Câu lệnh kết hợp

- Câu lệnh kết hợp được tạo thành từ một cặp dấu ngoặc nhọn {} và có thể có các khai báo và các câu lệnh nằm trong cặp dấu ngoặc nhọn đó.
- Câu lệnh kết hợp là một khối.
- Ví dụ:

```
vd5-1.c x
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     }
```

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

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

# Các câu lệnh rẽ nhánh

# Các câu lệnh rẽ nhánh

- Câu lệnh rẽ nhánh lựa chọn thực hiện câu lệnh trong 1 tập các câu lệnh tùy theo giá trị của biểu thức điều khiển:
- Có 3 định dạng sau (2 định dạng của câu lệnh if và câu lệnh switch):
  - if (*biểu thức*) *câu lệnh*
  - if (*biểu thức*) *câu lệnh* else *câu lệnh*
  - switch (*biểu thức*) *câu lệnh*
- Câu lệnh rẽ nhánh là một khối con của khối chứa nó, đồng thời mỗi câu lệnh thành phần cũng là một khối con của nó.

# Câu lệnh if

- Dạng 1 nhánh: *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.
- Dạng 2 nhánh: *if (biểu thức) câu lệnh 1 else câu lệnh 2*
  - Câu lệnh thành phần thứ nhất được thực hiện nếu biểu thức điều khiển có giá trị  $\neq 0$ , nếu ngược lại thì câu lệnh thành phần thứ 2 được thực hiện.



## Ví dụ 5.2. Câu lệnh if

```
1  #include <stdio.h>
2
3  int main() {
4      int n;
5      printf("Nhập 1 số nguyên dương: ");
6      scanf("%d", &n);
7      if (n <= 0) {
8          printf("%d không thỏa mãn yêu cầu.\n", n);
9          return 0;
10     }
11     if (n % 2 == 0) {
12         int sum = n * (n + 2) / 4;
13         printf("2 + ... + %d = %d\n", n, sum);
14     } else {
15         int sum = (n + 1) * (n + 1) / 4;
16         printf("1 + ... + %d = %d\n", n, sum);
17     }
18     return 0;
19 }
```

```
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
```

*Câu lệnh if  
1 nhánh*

*Câu lệnh if  
đầy đủ với  
2 nhánh.*

# Các câu lệnh if nhiều mức

- Nếu câu lệnh if có câu lệnh thành phần cũng là câu lệnh if và cứ tiếp tục như vậy tạo thành cấu trúc lựa chọn phức tạp với nhiều tầng điều kiện có thể khiến mã nguồn khó đọc và khó bảo trì.

- Ví dụ:

**Vấn đề:** Khó xác định nhánh else nào gắn với nhánh if nào.

**Giải pháp:** Chia khối và có thể tìm cách diễn đạt theo cách khác.

```
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
```

*Nhánh else được gắn với nhánh if gần nhất.*

## Ví dụ 5.3. Chia khối và các lệnh if nhiều mức

```
6 #include <stdio.h>
7
8 int main() {
9     printf("Nhập 2 số nguyên a và b: ");
10    int a, b;
11    scanf("%d%d", &a, &b);
12    if (a >= 0 && b >= 0) {
13        printf("Cùng dấu\n");
14    } else if (a < 0 && b < 0) {
15        printf("Cùng dấu\n");
16    } else {
17        printf("Khác dấu\n");
18    }
19    return 0;
20 }
```

*Chia khối giúp việc xác định các cặp if ... else.. dễ dàng hơn.*

```
6 #include <stdio.h>
7
8 int main() {
9     printf("Nhập 2 số nguyên a và b: ");
10    int a, b;
11    scanf("%d%d", &a, &b);
12    if (a >= 0) {
13        if (b < 0) {
14            printf("Khác dấu\n");
15        } else {
16            printf("Cùng dấu\n");
17        }
18    } else if (b < 0) {
19        printf("Cùng dấu\n");
20    } else {
21        printf("Khác dấu\n");
22    }
23    return 0;
24 }
```

# Câu lệnh switch: Yêu cầu và ý nghĩa

*Biểu thức điều khiển của câu lệnh switch phải có kiểu số nguyên. Quy tắc nâng kiểu số nguyên được áp dụng cho các giá trị nhãn của các case. Mỗi (nhãn) case phải có 1 giá trị nhãn duy nhất sau khi nâng kiểu. Đồng thời câu lệnh switch được có tối đa một nhãn default.*

- Câu lệnh switch chuyển điều khiển thực hiện lệnh tới nhãn case có giá trị nhãn bằng giá trị biểu thức điều khiển. Nếu không tồn tại nhãn case nào như vậy thì điều khiển thực hiện lệnh được chuyển tới nhãn default (nếu có, hoặc kết thúc thực hiện lệnh switch nếu không).
- Khi điều khiển được chuyển tới 1 nhãn thì phần mã nguồn (các khởi tạo và các câu lệnh) đứng trước nhãn đó trong phạm vi câu lệnh switch bị bỏ qua (không được thực hiện).

*(Tham khảo thêm về nhãn và câu lệnh goto)*



## Ví dụ 5.4. Rẽ nhánh với switch

```
6  #include <stdio.h>
7
8  int main() {
9      int n;
10     printf("Nhập 1 số nguyên trong khoảng [0,...,9]: ");
11     scanf("%d", &n);
12     switch (n) {
13         case 0: printf("Không\n");
14         case 1: printf("Một\n");
15         case 2: printf("Hai\n");
16         case 3: printf("Ba\n");
17         case 4: printf("Bốn\n");
18         case 5: printf("Năm\n");
19         case 6: printf("Sáu\n");
20         case 7: printf("Bảy\n");
21         case 8: printf("Tám\n");
22         case 9: printf("Chín\n");
23         default: printf("Ngoài khoảng\n");
24     }
25     return 0;
26 }
```

*Được bỏ qua khi nhập 5*

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

(Sẽ tiếp tục được cập nhật)

# Các câu lệnh lặp

# Các câu lệnh di chuyển



