

BIỂU THỨC TRONG NGÔN NGỮ C

huydq@soict.hust.edu.vn

Nội Dung Chính

- Biểu thức
- Toán tử

Biểu thức

- Kết hợp các giá trị bằng toán tử hoặc gọi hàm để tạo giá trị mới
- Giá trị trả về luôn có kiểu xác định
- Các toán tử có thể được phân loại
 - theo ngôi (1 hoặc 2 toán hạng)
 - theo chức năng (toán học, logic, ...)

Mục đích sử dụng

- Làm vế phải của lệnh gán.
- Làm toán hạng trong các biểu thức khác.
- Làm tham số thực sự trong lời gọi hàm.
- Làm biểu thức kiểm tra trong các cấu trúc điều khiển
 - Cấu trúc lặp: for, while, do while.
 - Cấu trúc rẽ nhánh: if, switch.

Các loại biểu thức

- Biểu thức số học
- Biểu thức quan hệ
- Biểu thức logic

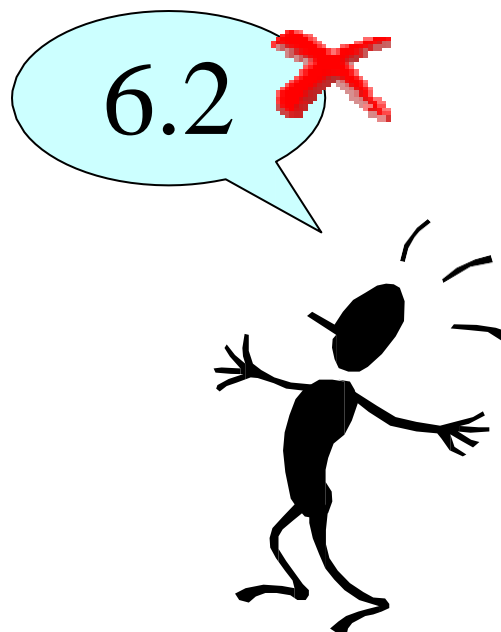
Biểu thức số học

- Là biểu thức mà giá trị của nó là các đại lượng số học (số nguyên, số thực).
 - Sử dụng các toán tử là các phép toán số học (+, -, *, /, ...)
 - Các toán hạng là các đại lượng số học (hằng số, biến, biểu thức khác).
- Chú ý: phép chia **số nguyên/số nguyên \rightarrow số nguyên**

Ví dụ biểu thức

$$1 + 2 * 3 - 4 / 5$$

$$= 1 + (2 * 3) - (4 / 5)$$



Ví dụ (tiếp)

$$1 + 2 * 3 - 4 / 5 =$$

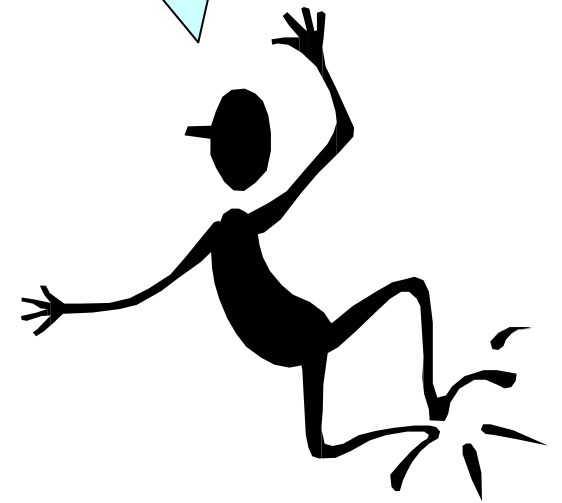
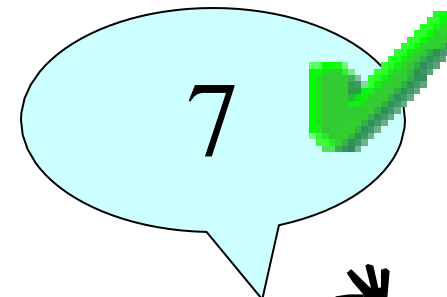
$$1 + (2 * 3) - (4 / 5)$$

Chia hai số nguyên cho
kết quả vẫn là một số
nguyên



Ví dụ (tiếp)

$$\begin{array}{l} 1 + 2 * 3 - 4 / 5 = \\ 1 + (2 * 3) - \underbrace{(4 / 5)} \\ = 0 \end{array}$$



Ví dụ (tiếp)

- Dùng số thực để tạo biểu thức có kết quả là số thực

$$\begin{aligned} & 1 + 2 * 3 - 4.0 / 5 \\ &= 1 + (2 * 3) - (4.0 / 5) \\ &= 1 + 6 - 0.8 \\ &= 6.2 \end{aligned}$$

Biểu thức quan hệ

- Là những biểu thức có sử dụng các toán tử quan hệ như lớn hơn, nhỏ hơn, khác nhau...
- Chỉ có thể trả về 1 trong 2 giá trị logic Đúng (TRUE) hoặc Sai (FALSE)
- Ví dụ

- $5 > 7$

- // có giá trị logic là sai, FALSE

- $9 \neq 10$

- // có giá trị logic là đúng, TRUE

- $2 \geq 2$

- // có giá trị logic là đúng, TRUE

- $a > b$

- // giả sử a, b là 2 biến kiểu int

Biểu thức logic

- Là biểu thức trả về các giá trị logic Đúng/Sai
 - Các phép toán logic gồm có
 - **AND** VÀ logic, sử dụng toán tử &&
 - **OR** HOẶC logic, sử dụng toán tử ||
 - **NOT** PHỦ ĐỊNH, sử dụng toán tử !
- Biểu thức quan hệ là trường hợp riêng của biểu thức logic.
- Biểu thức logic cũng trả về một giá trị số học 0/1

Biểu thức logic → Ví dụ

- `(5 > 7) && (9 != 10)` • // có giá trị logic là sai, FALSE
- `0 || 1` • // có giá trị logic là đúng, TRUE
- `(5 > 7) || (9 != 10)` • // có giá trị logic là đúng, TRUE
- `0` • // có giá trị logic là sai, FALSE
- `!0` • // phủ định của 0, có giá trị logic là đúng, TRUE
- `3` • // có giá trị logic là đúng, TRUE
- `!3` • // phủ định của 3, có giá trị logic là sai, FALSE
- `(a > b) && (a < b)` • // Có giá trị sai, FALSE. Giả sử a, b là 2 biến kiểu int

Dùng **int** như logic

- Trong C, các giá trị logic được biểu diễn bằng số nguyên
 - Giá trị nguyên 0 là sai.
 - Giá trị nguyên khác 0 là đúng (thường dùng số 1)
- Mọi biểu thức trong C đều trả về kết quả số
- Một biểu thức logic được đánh giá là đúng (true) sẽ trả về giá trị 1, ngược lại là 0.

Nội Dung Chính

- Biểu thức
- Toán tử

Các toán tử chính

- Toán tử số học
- Toán tử quan hệ
- Toán tử logic
- Toán tử logic bit
- Toán tử gán

Các toán tử số học

Toán tử	Ý nghĩa	Kiểu dữ liệu của toán hạng	Ví dụ (<i>int a = 12; float x=3.0</i>)
-	Đảo dấu	float, double, int, long,... (<i>Số nguyên hoặc thực</i>)	-12, -12.34, - a, - x
+	Cộng	float, double, int, long,...	12 + x
-	Trừ	float, double, int, long,...	12.0 - x
*	Nhân	float, double, int, long,... (<i>Số nguyên hoặc thực</i>)	12 * 3.0 12 * 3
/	Chia	Nếu có ít nhất 1 toán hạng là số thực	17.0/3.0 → 5.666667 17/3.0 → 5.666667 17.0/3 → 5.666667
/	Chia lấy thương	Số nguyên int, long,...	17/3 → 5
%	Chia lấy số dư	Số nguyên: int, long,...	17%3 → 2

Các toán tử quan hệ

$<$, $>$, $<=$, $>=$, $==$, $!=$

- Dùng cho phép so sánh giá trị 2 toán hạng
- Kết quả phép so sánh là một số nguyên
 - 1 nếu quan hệ có kết quả là đúng,
 - 0 nếu quan hệ có kết quả sai
- Ví dụ:
 - $6 > 4 \rightarrow$ Trả về giá trị 1
 - $6 < 4 \rightarrow$ Trả về giá trị 0
 - `int b = (x != y);`

Nếu x và y khác nhau, biểu thức đúng và b mang giá trị 1. Ngược lại biểu thức sai và b mang giá trị 0

Toán tử logic

- ...dùng để xây dựng các biểu thức logic
- Và (&&)
- Hoặc (||)
- Phủ định (!)
- So sánh bằng (==)
- So sánh khác (!=)
- So sánh lớn bé (<, >, <=, >=)

Các toán tử logic (tiếp)

- Và logic ($\&$) :
 - Cho kết quả đúng (trả về giá trị 1) khi cả 2 toán hạng đều đúng (khác 0)
 - Ví dụ: $3 < 5 \&\& 4 < 6 \rightarrow 1$; $3 < 5 \&\& 5 > 6 \rightarrow 0$
- Hoặc logic ($\|$):
 - Cho kết quả sai (trả về giá trị 0) chỉ khi cả 2 toán hạng đều sai (bằng 0)
 - Ví dụ: $4 \| 5 < 3 \rightarrow 1$; $5 < 5 \| 2 > 6 \rightarrow 0$
- Phủ định logic ($!$):
 - Cho kết quả đúng (1) hoặc sai (0) khi toán hạng là sai (0) hoặc đúng (khác 0)
 - Ví dụ: $!3 \rightarrow 0$; $!(2 > 3) \rightarrow 1$;

Ví dụ

Tránh nhầm
lẫn với và bit
(**&**)

Tránh nhầm
lẫn với hoặc
bit (**|**)

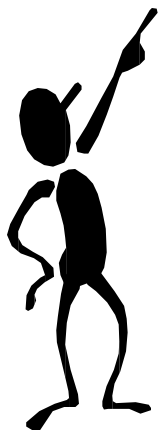
Tránh nhầm
lẫn với đảo
bit (**~**)

5 **&&** 4 → 1

1 **|** 4 → 1

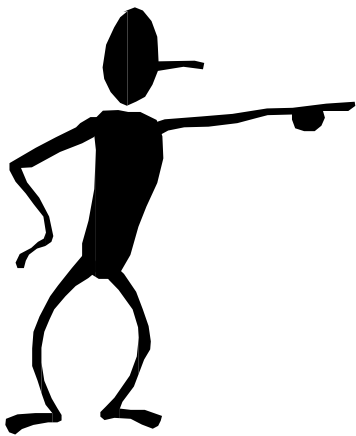
! 0 → 0

! 0 **|** 0 **&&** 2 → 1



“Ngắn mạch”

- Không nhất thiết phải sử dụng tất cả các toán hạng để đánh giá một biểu thức logic, chỉ cần đánh đến khi giá trị biểu thức được xác lập



1 || 2

0 || 1 || 2

1 && 0 && -1

Ví dụ toán tử so sánh

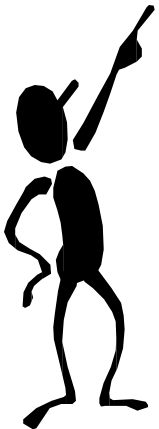
Tránh nhầm
lẫn với phép
gán (=)

3 == 4 → 0

3 != 4 → 1

3 < 4 → 1

3 < 4 && 5 > 2 → 1



Lỗi thường gặp

```
#include <stdio.h>

/* Loi thuong gap */

int main()
{
    int score;

    scanf("%d", &score);

    if ( score == 9 || 10 )
    {
        printf("Xuat sac\n");
    }
    return 0;
}
```

Giá trị trả về
luôn là 1

Giá trị trả về là
0 hoặc 1

Lỗi thường gặp (tiếp)

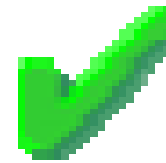
```
#include <stdio.h>

/* Chương trình dung */

int main()
{
    int score;

    scanf("%d", &score);

    if ( score == 9 || score == 10 )
    {
        printf("Xuat sac\n");
    }
    return 0;
}
```



Lỗi thường gặp (tiếp)

```
#include <stdio.h>

/* Loi thuong gap */

int main()
{
    int score;

    scanf("%d", &score);

    if ( 8 <= score <= 10 )
    {
        printf("Gioi\n");
    }
    return 0;
}
```

Giá trị trả về
luôn là 1

Giá trị trả về là
0 hoặc 1

Lỗi thường gặp (tiếp)

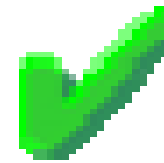
```
#include <stdio.h>

/* Chương trình dung */

int main()
{
    int score;

    scanf("%d", &score);

    if ( 8 <= score && score <= 10 )
    {
        printf("Gioi\n");
    }
    return 0;
}
```



Toán tử trên bit

- Toán tử trên bit (bitwise operator) được sử dụng với kiểu số nguyên

Và nhị phân:

$Op1 \& Op2$

Hoặc nhị phân :

$Op1 | Op2$

Hoặc có loại trừ nhị phân:

$Op1 \wedge Op2$

Đảo bit nhị phân :

$\sim Op$

Dịch trái n bit:

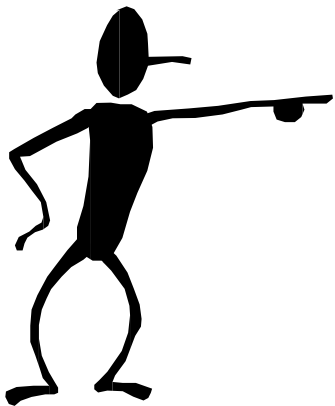
$Op \ll n$

Dịch phải n bit:

$Op \gg n$

Toán tử trên bit

- Một biểu thức chỉ sử dụng các toán tử trên bit không phải là biểu thức logic.
- Kết quả của biểu thức này là một số nguyên bất kì



$$5 \ \& \ 4 \rightarrow 4$$

$$1 \ | \ 4 \rightarrow 5$$

$$\sim 0 \rightarrow 0\text{xFFFF}$$

Toán tử trên bit (tiếp)

char Op1 = 83, Op2 = -38, Op = 3;

char r = Op1 & Op2;

0 1 0 1 0 0 1 1

1 1 0 1 1 0 1 0

r = 0 1 0 1 0 0 1 0 → (82)

char r = Op1 | Op2;

0 1 0 1 0 0 1 1

1 1 0 1 1 0 1 0

r = 1 1 0 1 1 0 1 1 → (-37)

char r = Op1 ^ Op2;

0 1 0 1 0 0 1 1

1 1 0 1 1 0 1 0

r = 1 0 0 0 1 0 0 1 → (-119)

char r = ~ Op2;

1 1 0 1 1 0 1 0

r = 0 0 1 0 0 1 0 1 → (37)

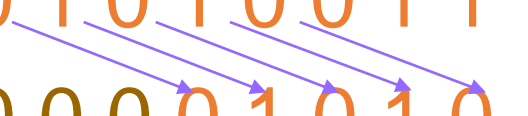
unsigned char r = Op1 | Op2; r = 1 1 0 1 1 0 1 1 → 219

Toán tử trên bit (tiếp)

char Op1 = 83, Op2 = -38, Op = 3;

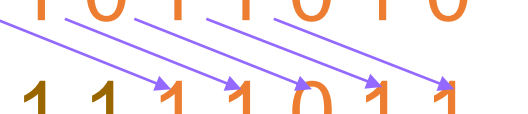
char r = Op1 >> Op;

0 1 0 1 0 0 1 1
r = 0 0 0 0 1 0 1 0 → (10)



char r = Op2 >> Op;

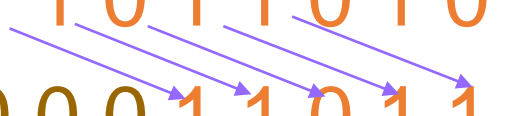
1 1 0 1 1 0 1 0
r = 1 1 1 1 1 0 1 1 → (-5)



unsigned char Op = 218;

unsigned char r = Op >> 3;

1 1 0 1 1 0 1 0
r = 0 0 0 1 1 0 1 1 → (27)



char r = Op2 << 2;

r = 0 1 1 0 1 0 0 0 → (104)

(unsigned) int r = Op2 << 2 → ?

Toán tử gán

$\text{Biến} = \text{Biểu_thức};$

- Là toán tử được sử dụng thường xuyên
 - Biểu thức bên phải dấu bằng được tính toán
 - Giá trị của biểu_thức được gán cho biến ở vế trái
- Ví dụ:
 - `int a, b, c;`
 - `a = 3;`
 - `b = a + 5;`
 - `c = a * b;`

Toán tử gán

- Gán (=) cũng là một toán tử. Vì vậy sử dụng toán tử này tạo ra một biểu thức và trả về giá trị
- Kết quả của biểu thức gán là giá trị bên phải của biểu thức
- Ví dụ:
 $(x = 4) \rightarrow 4$
 $(y = 0) \rightarrow 0$
- Có thể tạo 1 biểu thức với một chuỗi toán tử gán
 $x = y = z = 4$

Lỗi thường gặp (tiếp)

```
#include <stdio.h>

/* Loi gap nhieu trong C */

int main()
{
    int score;

    scanf("%d", &score);

    if (score = 9 || score = 10)
    {
        printf("Gioi!\n");
    }
    return 0;
}
```

Nhầm với
phép gán

Lỗi thường gặp (tiếp)

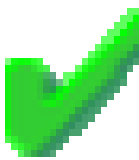
```
#include <stdio.h>

/* Probably the most common C error. */

int main()
{
    int score;

    scanf("%d", &score);

    if (score == 9 || score == 10)
    {
        printf("OK!\n");
    }
    return 0;
}
```



Một số toán tử gán mở rộng

Toán tử	Ví dụ	Biểu thức tương đương
$+=$	$x += 5$	$x = x + 5$
$-=$	$x -= 5$	$x = x - 5$
$*=$	$x *= 5$	$x = x * 5$
$/=$	$x /= 5$	$x = x / 5$
$\%=$	$x \% = 5$	$x = x \% 5$

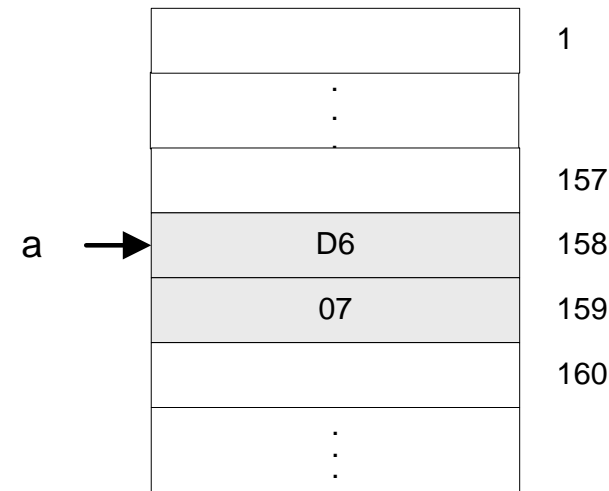
Toán tử tăng giảm

- **++** là toán tử tăng
 ++i tương đương với **i = i + 1**
- **--** là toán tử giảm
 --j tương đương với **j = j - 1**
- Có 2 dạng viết: tiền tố (**++i**) và hậu tố (**i++**)
- Chúng khác nhau ở giá trị trả về của biểu thức.
Ví dụ nếu **i = 5**
 - Tiền tố trả về giá trị sau khi đã cộng, **(++i) → 6**
 - Hậu tố trả về giá trị trước khi cộng, **(i++) → 5**
 - Trong cả hai trường hợp giá trị của i đều tăng lên 1

Toán tử lấy địa chỉ

&Tên_biến

- Biến thực chất là một vùng nhớ của máy tính được đặt tên → tên của biến
- Mọi ô nhớ trên bộ nhớ máy tính đều được đánh địa chỉ.
→ Mọi biến đều có địa chỉ



- Ví dụ:

short int a = 2006;

&a là địa chỉ của ô nhớ chứa giá trị biến a

Toán tử điều kiện

- Có thể viết một biểu thức mà giá trị của nó phụ thuộc vào một điều kiện

<Điều kiện> ? <Biểu thức 1> : <Biểu thức 2>

– Viết biểu thức tính max của a và b

```
int max, a, b;
```

...

```
max = ( a > b ) ? a : b;
```

Ép kiểu

- **Phép gán** chỉ thực hiện trên **biến** và **giá trị** có **cùng kiểu**
- C chuyển kiểu tự động cho phép gán nếu sự chuyển kiểu đó không làm mất thông tin.

char → **int** → **long int** → **float** → **double** → **long double**

- Ví dụ chuyển từ **int** về **float**

```
int a;
```

```
float f;
```

```
f = a; /* OK */
```

```
a = f; /* KO */
```

- Trong trường hợp bị mất thông tin ta phải thực hiện ép kiểu. Ví dụ nếu chuyển từ **float** về **int**

```
a = (int) f;
```


Thứ tự ưu tiên các toán tử

Mức	Toán tử	Chức năng	Chiều	Chiều kết hợp với các toán hạng
1	-> . [] () ++ _{hậu tố} -- _{hậu tố}	Lựa chọn, chỉ số...	→	
2	++ -- ~ ! + - * & () sizeof	Toán tử 1 ngôi, ép kiểu,...	←	
3	* / %	Toán tử số học lớp nhân	→	
4	+ -	Toán tử số học lớp cộng	→	
5	>> <<	Dịch bit	→	
6	< <= > >=	Toán tử quan hệ	→	
7	== !=	Bằng, khác	→	
8	&	AND nhị phân	→	
9	^	XOR nhị phân	→	
10		OR nhị phân	→	
11	&&	AND logic	→	
12		OR logic	→	
13	? :	Toán tử phỏng điều kiện	←	
14	= *= += <=& &= ...	Toán tử gán	←	

Thứ tự ưu tiên các toán tử

- Nguyên tắc
- Biểu thức con trong ngoặc được tính toán trước
- Phép toán một ngôi đứng bên trái toán hạng được kết hợp với toán hạng đi liền nó.
- Toán hạng đứng cạnh hai toán tử
 - Nếu hai toán tử có độ ưu tiên khác nhau thì toán tử nào có độ ưu tiên cao hơn sẽ kết hợp với toán hạng
 - Nếu hai toán tử cùng độ ưu tiên thì dựa vào trật tự kết hợp của các toán tử để xác định toán tử được kết hợp với toán hạng.
- Ví dụ
$$a < 10 \ \&\& \ 2 * b < c \equiv (a < 10) \ \&\& \ ((2 * b) < c)$$

Chú ý: `int x = 5, a = 5 * x++;` \rightarrow `a = 25, x = 6`

Ví dụ

```
const int N=10;  
float S= 0.0;  
int b;  
S = N/3 +1;  
b=(S>4);
```

S= ? b = ?

```
int a= 3, b=4, c;  
c = a++ * ++b;
```

a= ? b= ? c= ?

```
int k ,num=30;  
k = num>5 ? (num <=10 ? 100 : 200) : 500;  
k=?
```

Ví dụ

```
const int N=10;  
float S= 0.0;  
int b;  
S = N/3 +1;  
b=(S>4);
```

S= 4 b = 0

```
int a= 3, b=4, c;  
c = a++ * ++b;
```

a=4 b=5 c=15

```
int k ,num=30;  
k = num>5 ? (num <=10 ? 100 : 200) : 500;  
k=200
```

Ví dụ

$$7+5 \&\& 4 < 2+3-2/3 \mid \mid 5 > 2+1$$

$$(7+5) \&\& 4 < 2+3- (2/3) \mid \mid 5 > (2+1)$$

$$12 \&\& 4 < (2+3-0) \mid \mid (5 > 3)$$

$$12 \&\& (4 < 5) \mid \mid 1$$

$$(12 \&\& 1) \mid \mid 1$$

$$1 \mid \mid 1$$

$$= 1$$