

# Bài 13: TOÁN TỬ QUAN HỆ, LOGIC, BITWISE, MISC VÀ ĐỘ ƯU TIÊN TOÁN TỬ TRONG C++

Xem bài học trên website để ủng hộ Kteam: [Toán tử quan hệ, logic, bitwise, misc và độ ưu tiên toán tử trong C++](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

## Dẫn nhập

Ở bài học trước, bạn đã nắm được [TOÁN TỬ SỐ HỌC, TOÁN TỬ TĂNG GIẢM VÀ TOÁN TỬ GÁN TRONG C++](#).

Hôm nay, mình sẽ hướng dẫn về **Toán tử quan hệ, logic, bitwise, misc và độ ưu tiên toán tử trong C++ (Operators)**.

---

## Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- [BIẾN TRONG C++ \(Variables\)](#)
- [CÁC KIỂU DỮ LIỆU CƠ BẢN TRONG C++ \(Integer, Floating point, Character, Boolean\)](#)
- [NHẬP VÀ XUẤT DỮ LIỆU TRONG C++ \(Input and Output\)](#)

- TOÁN TỬ SỐ HỌC, TOÁN TỬ TĂNG GIẢM VÀ TOÁN TỬ GÁN TRONG C++

Trong bài ta sẽ cùng tìm hiểu các vấn đề:

- Toán tử quan hệ trong C++
- Toán tử logic trong C++
- Toán tử trên bit trong C++
- Các toán tử hỗn hợp trong C++
- Độ ưu tiên toán tử trong C++

## Toán tử quan hệ trong C++ (Relational operators)

Toán tử quan hệ dùng để **so sánh 2 toán hạng** với nhau. Sẽ trả về 2 giá trị là **1 (true)** hoặc **0 (false)**.

Bảng bên dưới mô tả các toán tử quan hệ trong C++, giả sử **x = 6, y = 9**

Operator	Symbol	Form	Operation	Example
Greater than	>	$x > y$	true if x is greater than y, false otherwise	$(x > y)$ is false
Less than	<	$x < y$	true if x is less than y, false otherwise	$(x < y)$ is true
Greater than or equals	>=	$x >= y$	true if x is greater than or equal to y, false otherwise	$(x >= y)$ is false
Less than or equals	<=	$x <= y$	true if x is less than or equal to y, false otherwise	$(x <= y)$ is true
Equality	==	$x == y$	true if x equals y, false otherwise	$(x == y)$ is false
Inequality	!=	$x != y$	true if x does not equal y, false otherwise	$(x != y)$ is true

**Chú ý:** Phân biệt toán tử **gán bằng (=)** và toán tử **so sánh bằng (==)**.

**Ví dụ:**

```
#include <iostream>
using namespace std;

int sum(int a, int b)
{
    return a + b;
}

int main()
{
    cout << "Enter an integer: ";
    int x;
    cin >> x;

    cout << "Enter another integer: ";
    int y;
    cin >> y;

    if (x == y)
        cout << x << " == " << y << "\n";
    if (x != y)
        cout << x << " != " << y << "\n";
    if (x > y)
        cout << x << " > " << y << "\n";
    if (x < y)
        cout << x << " < " << y << "\n";
    if (x >= y)
        cout << x << " >= " << y << "\n";
    if (x <= y)
        cout << x << " <= " << y << "\n";

    return 0;
}
```

**Outputs:**



## Toán tử quan hệ và so sánh số chấm động?

Trong lập trình, việc **so sánh trực tiếp 2 số chấm động** là điều không nên và có thể cho ra những kết quả không mong muốn. Đó là do lỗi làm tròn của số chấm động, vấn đề này đã được giải thích trong bài [SỐ TỰ NHIÊN & SỐ CHẤM ĐỘNG TRONG C++](#) (Integer, Floating point).

### Ví dụ:

```
#include <iostream>
#include <iomanip>      // for std::setprecision()
using namespace std;

int main()
{
    double d1{ 1.0 };
    double d2{ 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 };

    if (d1 == d2)
        cout << "d1 == d2" << "\n";
    else if (d1 > d2)
        cout << "d1 > d2" << "\n";
    else if (d1 < d2)
        cout << "d1 < d2" << "\n";

    cout << std::setprecision(20);    // show 20 digits
    cout << "d1 = " << d1 << endl;
    cout << "d2 = " << d2 << endl;
```

```
    return 0;  
}
```

### Outputs:



Trong chương trình trên, trong toán học thì 2 biến **d1 == d2**, nhưng trong lập trình biến **d1 > d2** vì lỗi làm tròn số dấu chấm động.

Tương tự, bạn hãy thử với trường hợp **0.1 + 0.7 == 0.8 ?**

**Chú ý:** Không bao giờ so sánh hai giá trị dấu chấm động bằng nhau hay không. Hầu như luôn luôn có sự khác biệt nhỏ giữa hai số chấm động. Cách phổ biến để so sánh 2 số chấm động là tính khoảng cách giữa 2 số đó, nếu khoảng cách đó là rất nhỏ thì ta cho là bằng nhau. Giá trị dùng để so sánh với khoảng cách đó thường được gọi là **epsilon**.

## Toán tử logic trong C++ (Logical operators)

Nếu chỉ sử dụng toán tử quan hệ để so sánh biểu thức quan hệ đúng hay sai, bạn chỉ có thể kiểm tra một điều kiện tại một thời điểm. Nhưng thực tế, có lúc bạn sẽ cần kiểm tra nhiều điều kiện cùng lúc.

**Ví dụ:** để trở thành một soái ca thì bạn phải có nhiều điều kiện như cầm, kỳ, thi, họa. Lúc này không chỉ đơn giản 1 điều kiện nữa.

**Toán tử logic (Logical operators)** sẽ kiểm tra nhiều điều kiện cùng lúc giúp bạn. Có 3 toán tử logic trong C++

Operator	Symbol	Form	Operation
<b>Logical NOT</b>	!	!x	true if x is false, or false if x is true
<b>Logical AND</b>	&&	x && y	true if both x and y are true, false otherwise
<b>Logical OR</b>		x    y	true if either x or y are true, false otherwise

**Chú ý:** Luôn sử dụng dấu ngoặc đơn () khi thực hiện với các mệnh đề quan hệ để thể hiện rõ ràng ý nghĩa dòng lệnh và hạn chế sai sót. Với cách này, bạn thậm chí không cần nhớ nhiều về độ ưu tiên toán tử.

**Ví dụ:**

```
#include <iostream>
#include <iomanip>      // for std::setprecision()
using namespace std;

int main()
{
    cout << "Enter a number: ";
    int value;
    cin >> value;

    if (!value)
        cout << value << " is false" << endl;
```

```
else
    cout << value << " is true" << endl;

if ((value > 1) && (value < 100))
    cout << value << " is between 1 and 100" << endl;
else
    cout << value << " is not between 1 and 100" << endl;

if ((value == 1) || (value == 100))
    cout << value << " == 1 or " << value << " == 100" << endl;
else
    cout << value << " != 1 or " << value << " != 100" << endl;

return 0;
}
```

### Outputs:



## Toán tử trên bit trong C++ (Bitwise operators)

Toán tử trên bit dùng để thao tác với các bit trên một biến.

**Tại sao cần thao tác trên bit?** Trong quá khứ, bộ nhớ máy tính chưa phát triển, vấn đề về quản lý bộ nhớ là rất quan trọng. Vì vậy, người lập trình cần tận dụng tối đa các bit trong bộ nhớ.

**Ví dụ:** Các biến được lưu trong bộ nhớ ở một địa chỉ duy nhất, những địa chỉ này được xác định với đơn vị nhỏ nhất là byte. **Xét kiểu dữ liệu `bool`**, nó chỉ nắm **giữ 2 giá trị `true (1)` hoặc `false (0)`**. Kiểu `bool` chỉ cần 1 bit để lưu trữ dữ liệu, nhưng nó lại chiếm giữ 1 byte trong bộ nhớ, vậy 7 bit còn lại sẽ là lãng phí. Sử dụng toán tử trên bit giúp bạn có thể chứa 8 biến kiểu `bool` vào 1 byte duy nhất, và tiết kiệm bộ nhớ đáng kể.

Trong quá khứ, sử dụng toán tử trên bit rất được ưu chuộng. Ngày nay, bộ nhớ máy tính đã phát triển và rẻ hơn, lập trình viên thường **quan tâm đến tính dễ hiểu và dễ nâng cấp của mã nguồn**. Do đó, việc **thao tác trên bit không còn được ưu chuộng**, ngoại trừ những trường hợp cần tối ưu tối đa tốc độ, bộ nhớ (những chương trình thao tác big data, những game lớn, lập trình nhúng ...).

Vì các toán tử trên bit ít gặp nên mình chỉ giới thiệu qua cho các bạn tham khảo, bạn có thể tự tìm hiểu thêm nếu muốn chuyên sâu hơn.

Bảng bên dưới gồm 6 toán tử thao tác trên bit

Operator	Symbol	Form	Operation
<b>left shift</b>	<code>&lt;&lt;</code>	<code>x &lt;&lt; y</code>	all bits in x shifted left y bits
<b>right shift</b>	<code>&gt;&gt;</code>	<code>x &gt;&gt; y</code>	all bits in x shifted right y bits
<b>bitwise NOT</b>	<code>~</code>	<code>~x</code>	all bits in x flipped
<b>bitwise AND</b>	<code>&amp;</code>	<code>x &amp; y</code>	each bit in x AND each bit in y
<b>bitwise OR</b>	<code> </code>	<code>x   y</code>	each bit in x OR each bit in y
<b>bitwise XOR</b>	<code>^</code>	<code>x ^ y</code>	each bit in x XOR each bit in y

Bảng bên dưới gồm 5 toán tử gán trên bit



Operator	Symbol	Form	Operation
Left shift assignment	<<=	x <<= y	Shift x left by y bits
Right shift assignment	>>=	x >>= y	Shift x right by y bits
Bitwise OR assignment	=	x  = y	Assign x   y to x
Bitwise AND assignment	&=	x &= y	Assign x & y to x
Bitwise XOR assignment	^=	x ^= y	Assign x ^ y to x

## Các toán tử hỗn hợp trong C++ (Misc Operators)

### Sizeof operator

Operator	Symbol	Form	Operation
Sizeof	sizeof	sizeof(type)	Returns size of type or variable in bytes
		sizeof(variable)	

Để xác định kích thước của một kiểu dữ liệu trên một máy tính cụ thể, C++ cung cấp cho bạn toán tử **sizeof**. **Toán tử sizeof** là toán tử một ngôi, **nhận vào một kiểu dữ liệu hoặc một biến**, và **trả về kích thước (byte)** của kiểu dữ liệu hoặc biến đó. Toán tử này sizeof đã được giải thích chi tiết trong bài Số tự nhiên và Số chấm động trong C++ (Integer, Floating point).

## Comma operator

Operator	Symbol	Form	Operation
Comma	,	x, y	Evaluate x then y, returns value of y

Các biểu thức đặt cách nhau bằng dấu phẩy (,), các biểu thức con lần lượt được tính từ trái

sang phải. Biểu thức mới nhận được là giá trị của biểu thức bên phải cùng.

### Ví dụ:

```
x = (a++, b = b + 2);  
  
// Tương đương với  
a++; b = b + 2; x = b;
```

### hoặc

```
int x = 0;  
int y = 2;  
int z = (++x, ++y); // increment x and y  
  
// Tương đương với  
int x = 0;  
int y = 2;  
++x;  
++y;  
int z = y;
```

**Chú ý:** Dấu phẩy có ưu tiên thấp nhất trong tất cả các toán tử.

Vì vậy, hai dòng code sau đây sẽ cho kết quả khác nhau:

```
z = (a, b); // z = b
z = a, b; // z = a, và b bị bỏ qua
```

**Chú ý:** Tránh sử dụng các toán tử dấu phẩy (,), ngoại trừ trường hợp dùng trong vòng lặp. Vấn đề về vòng lặp sẽ được hướng dẫn chi tiết trong bài [VÒNG LẶP FOR TRONG C++ \(For statements\)](#).

## Conditional operator

Operator	Symbol	Form	Operation
<b>Conditional</b>	?:	c ? x : y	If c is nonzero (true) then evaluate x, otherwise evaluate y

Toán tử điều kiện (?:) là toán tử 3 ngôi duy nhất trong C++ (có 3 toán hạng), tương đương với câu điều kiện (if/else statements).

### Cấu trúc câu điều kiện if/else:

```
if (condition) // nếu condition là true
    expression1; // thực thi câu lệnh này
else
    expression2; // nếu condition là false, thực thi câu lệnh này
```

### Hoặc :

```
if (condition) // nếu condition là true
    x = value1; // x = value 1
else
    x = value2; // nếu condition là false, x = value 2
```

**Viết lại dưới dạng toán tử điều kiện ( ?: ):**

```
(condition) ? expression1 : expression2;
```

**Hoặc:**

```
x = (condition) ? value1 : value2;
```

**Ví dụ:**

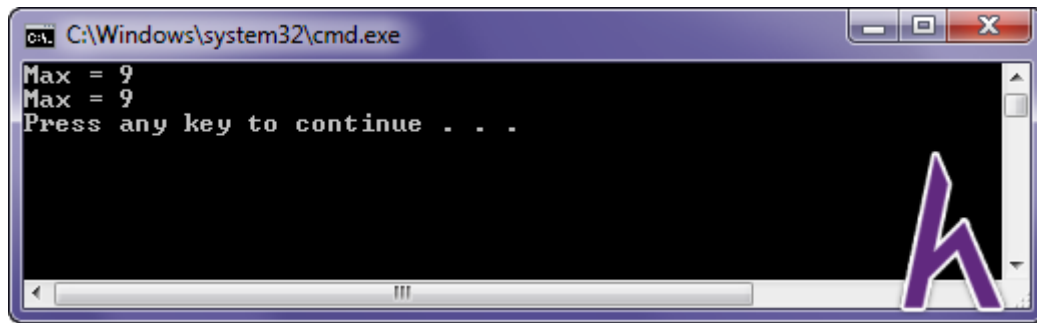
```
#include <iostream>
using namespace std;

int main()
{
    int x{ 6 }, y{ 9 }, max;
    if (x > y)
    {
        max = x;
    }
    else
    {
        max = y;
    }
    cout << "Max = " << max << endl;

    // Tương đương với
    max = (x > y) ? x : y;
    cout << "Max = " << max << endl;

    return 0;
}
```

**Outputs:**



**Chú ý:** Chỉ sử dụng toán tử điều kiện với những câu điều kiện đơn giản.

### Ví dụ:

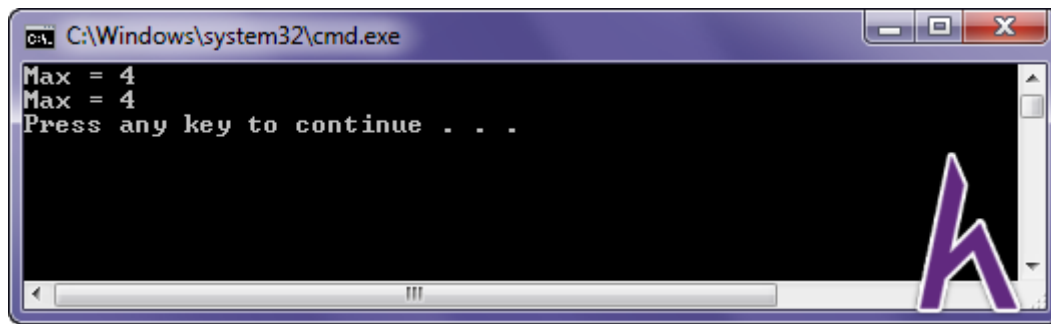
```
#include <iostream>
using namespace std;

int main()
{
    int a{ 3 }, b{ 2 }, c{ 4 }, max;

    // Khó hiểu, dễ sai => Không nên
    max = a > b ? (a > c ? a : c) : (b > c ? b : c);
    cout << "Max = " << max << endl;

    // Dễ hiểu => Nên
    max = a;
    if (max < b)
    {
        max = b;
    }
    if (max < c)
    {
        max = c;
    }
    cout << "Max = " << max << endl;

    return 0;
}
```

**Outputs:**

**Chú ý:** Toán tử điều kiện ( ?: ) có thể là một biểu thức (expression), trong khi câu điều kiện ( if/else ) chỉ là một câu lệnh (statements).

**Ví dụ:**

```
bool blsVip = true;  
  
// Initializing a const variable  
const double dPrice = blsVip ? 1000 : 500;
```

Trong ví dụ trên, không thể dùng câu điều kiện ( if/else ) để thay thế. Vì một hằng số phải được khởi tạo giá trị tại thời điểm khai báo.

## Một số toán tử khác

Operator	Description
. (dot) and -> (arrow)	<b>Member operators</b> are used to reference individual members of classes, structures, and unions.
Cast	<b>Casting operators</b> convert one data type to another. For example, <code>int(2.2000)</code> would return 2.
&	<b>Pointer operator &amp;</b> returns the address of an variable. For example <code>&amp;a;</code> will give actual address of the variable.
*	<b>Pointer operator *</b> is pointer to a variable. For example <code>*var;</code> will pointer to a variable var.

## Độ ưu tiên và quy tắc kết hợp toán tử trong C++

Để đánh giá đúng một biểu thức như  $6 + 9 * 8$ , bạn cần **hiểu rõ ý nghĩa mỗi toán tử** trong biểu thức đó, và **thứ tự thực hiện** của nó. **Thứ tự thực hiện của các toán tử** trong 1 biểu thức gọi là **độ ưu tiên của toán tử (operator precedence)**.

Khi áp dụng độ ưu tiên toán tử, biểu thức bên trên sẽ tương đương với  $6 + (9 * 8) = 78$ . Phép nhân (\*) có độ ưu tiên cao hơn phép cộng (+), compiler sẽ tự động hiểu và thực hiện đúng theo độ ưu tiên của từng toán tử.

Khi 2 toán tử có cùng độ ưu tiên trong 1 biểu thức, các **quy tắc kết hợp (associativity rules)** sẽ nói cho compiler biết nên thực hiện từ **trái sang phải (L->R)** hay từ **phải sang trái (R->L)**.

**Ví dụ:** bạn có biểu thức  $6 * 9 / 8$ . Trong biểu thức này, phép nhân (\*) và chia (/) cùng độ ưu tiên, nhưng nó có quy tắc kết hợp từ trái sang phải (L->R). Vì vậy nó sẽ tương đương  $(6 * 9) / 8$ .

Bảng độ ưu tiên (**operator precedence**) và quy tắc kết hợp (**associativity rules**) các toán tử trong C++

Prec/Ass	Operator	Description	Pattern
1 None	::	Global scope (unary)	::name
	::	Class scope (binary)	class_name::member_name

2 L->R	()	Parentheses	(expression)
	()	Function call	function_name(parameters)
	()	Initialization	type name(expression)
	{}	Uniform initialization (C++11)	type name{expression}
	type()	Functional cast	new_type(expression)
	static_cast<type>()	Functional cast (C++11)	new_type(expression)
	[]	Array subscript	pointer[expression]
	.	Member access from object	object.member_name
	->	Member access from object ptr	object_pointer->member_name
	++	Post-increment	lvalue++
	--	Post-decrement	lvalue--
	typeid	Run-time type information	typeid(type) or typeid(expression)
	const_cast	Cast away const	const_cast<type>(expression)
	dynamic_cast	Run-time type-checked cast	dynamic_cast<type>(expression)
	reinterpret_cast	Cast one type to another	reinterpret_cast<type>(expression)
	static_cast	Compile-time type-checked cast	static_cast<type>(expression)

3 R->L	+	Unary plus	+expression
	-	Unary minus	-expression
	++	Pre-increment	++lvalue
	--	Pre-decrement	--lvalue
	!	Logical NOT	!expression
	~	Bitwise NOT	~expression
	(type)	C-style cast	(new_type)expression
	sizeof	Size in bytes	sizeof(type) or sizeof(expression)
	&	Address of	&lvalue
	*	Dereference	*expression
	new	Dynamic memory allocation	new type
	new[]	Dynamic array allocation	new type[expression]
	delete	Dynamic memory deletion	delete pointer
	delete[]	Dynamic array deletion	delete[] pointer



<b>4 L-&gt;R</b>	->*	Member pointer selector	object_pointer->*pointer_to_member
	.*	Member object selector	object.*pointer_to_member
<b>5 L-&gt;R</b>	*	Multiplication	expression * expression
	/	Division	expression / expression
	%	Modulus	expression % expression
<b>6 L-&gt;R</b>	+	Addition	expression + expression
	-	Subtraction	expression - expression
<b>7 L-&gt;R</b>	<<	Bitwise shift left	expression << expression
	>>	Bitwise shift right	expression >> expression
<b>8 L-&gt;R</b>	<	Comparison less than	expression < expression
	<=	Comparison less than or equals	expression <= expression
	>	Comparison greater than	expression > expression
	>=	Comparison greater than or equals	expression >= expression
<b>9 L-&gt;R</b>	==	Equality	expression == expression
	!=	Inequality	expression != expression
<b>10 L-&gt;R</b>	&	Bitwise AND	expression & expression

<b>11 L-&gt;R</b>	^	Bitwise XOR	expression ^ expression
<b>12 L-&gt;R</b>		Bitwise OR	expression   expression
<b>13 L-&gt;R</b>	&&	Logical AND	expression && expression
<b>14 L-&gt;R</b>		Logical OR	expression    expression

<b>15 R-&gt;L</b>	?:	Conditional (see note below)	expression ? expression : expression
	=	Assignment	lvalue = expression
	*=	Multiplication assignment	lvalue *= expression
	/=	Division assignment	lvalue /= expression
	%=	Modulus assignment	lvalue %= expression
	+=	Addition assignment	lvalue += expression
	-=	Subtraction assignment	lvalue -= expression
	<<=	Bitwise shift left assignment	lvalue <<= expression
	>>=	Bitwise shift right assignment	lvalue >>= expression
	&=	Bitwise AND assignment	lvalue &= expression
	=	Bitwise OR assignment	lvalue  = expression
	^=	Bitwise XOR assignment	lvalue ^= expression
<b>16 R-&gt;L</b>	throw	Throw expression	throw expression
<b>17 L-&gt;R</b>	,	Comma operator	expression, expression

Có thể bạn sẽ không hiểu phần lớn những toán tử trong bảng trên ở thời điểm này, bạn chỉ cần quan tâm đến những toán tử vừa học ở phần trên. Những toán tử còn lại bạn có thể tự tìm hiểu nếu bạn có nhu cầu đặt biệt nào đó.

# Kết luận

Qua bài học này, bạn đã nắm được [Toán tử quan hệ, logic, bitwise, misc và độ ưu tiên toán tử trong C++ \(Operators\)](#).

Ở bài tiếp theo, bạn sẽ được học về [CƠ BẢN VỀ CHUỖI KÝ TỰ TRONG C++ \(String\)](#), là tiền đề để bạn có thể giải được các bài toán trong lập trình.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.

