Chương 2. Biểu thức

Chương này giới thiệu các toán tử xây dựng sẵn cho việc soạn thảo các biểu thức. Một biểu thức là bất kỳ sự tính toán nào mà cho ra một giá trị.

Khi thảo luận về các biểu thức, chúng ta thường sử dụng thuật ngữ **ước lượng.** Ví dụ, chúng ta nói rằng một biểu thức ước lượng một giá trị nào đó. Thường thì giá trị sau cùng chỉ là lý do cho việc ước lượng biểu thức. Tuy nhiên, trong một vài trường hợp, biểu thức cũng có thể cho các kết quả phụ. Các kết quả này là sự thay đổi lâu dài trong trạng thái của chương trình. Trong trường hợp này, các biểu thức C++ thì khác với các biểu thức toán học.

C++ cung cấp các toán tử cho việc soạn thảo các biểu thức toán học, quan hệ, luận lý, trên bit, và điều kiện. Nó cũng cung cấp các toán tử cho ra các kết quả phụ hữu dụng như là gán, tăng, và giảm. Chúng ta sẽ xem xét lần lượt từng loại toán tử. Chúng ta cũng sẽ thảo luận về các luật ưu tiên mà ảnh hưởng đến thứ tự ước lượng của các toán tử trong một biểu thức có nhiều toán tử.

2.1. Toán tử toán học

C++ cung cấp 5 toán tử toán học cơ bản. Chúng được tổng kết trong Bảng 2.1.

Bảng 2.1 Các toán tử toán học.

Toán tử	Tên	Ví dụ
+	Cộng	12 + 4.9 // cho 16.9
-	Trừ	3.98 - 4 // cho -0.02
*	Nhân	2 * 3.4 // cho 6.8
/	Chia	9 / 2.0 // cho 4.5
%	Lấy phần dư	13 % 3 // cho 1

Ngoại trừ toán tử lấy phần dư (%) thì tất cả các toán tử toán học có thể chấp nhận pha trộn các toán hạng số nguyên và toán hạng số thực. Thông thường, nếu cả hai toán hạng là số nguyên sau đó kết quả sẽ là một số

nguyên. Tuy nhiên, một hoặc cả hai toán hạng là số thực thì sau đó kết quả sẽ là một số thực (real hay double).

Khi cả hai toán hạng của toán tử chia là số nguyên thì sau đó phép chia được thực hiện như là một phép chia số nguyên và không phải là phép chia thông thường mà chúng ta sử dụng. Phép chia số nguyên luôn cho kết quả nguyên (có nghĩa là luôn được làm tròn). Ví dụ:

```
9 / 2       // được 4, không phải là 4.5!
-9 / 2      // được -5, không phải là -4!
```

Các phép chia số nguyên không xác định thường là các lỗi lập trình chung. Để thu được một phép chia số thực khi cả hai toán hạng là số nguyên, bạn cần ép một trong hai số nguyên về số thực:

```
int     cost = 100;
int     volume = 80;
double  unitPrice = cost / (double) volume;     // được 1.25
```

Toán tử lấy phần dư (%) yêu cầu cả hai toán hạng là số nguyên. Nó trả về phần dư còn lại của phép chia. Ví dụ 13%3 được tính toán bằng cách chia số nguyên 13 đi 3 để được 4 và phần dư là 1; vì thế kết quả là 1.

Có thể có trường hợp một kết quả của một phép toán toán học quá lớn để lưu trữ trong một biến nào đó. Trường hợp này được gọi là **tràn**. Hậu quả của tràn là phụ thuộc vào máy vì thế nó không được định nghĩa.Ví dụ:

```
unsigned char   k = 10 * 92;        // tràn: 920 > 255
```

Chia một số cho 0 là hoàn toàn không đúng luật. Kết quả của phép chia này là một lỗi run-time gọi là lỗi *division-by-zero* thường làm cho chương trình kết thúc.

2.2. Toán tử quan hệ

C++ cung cấp 6 toán tử quan hệ để so sánh các số. Các toán tử này được tổng kết trong Bảng 2.2. Các toán tử quan hệ ước lượng về 1 (thay cho kết quả đúng) hoặc 0 (thay cho kết quả sai).

Bảng 2.2 Các toán tử quan hệ.

Toán tử	Tên	Ví dụ
==	So sánh bằng	5 == 5 // cho 1
!=	So sánh không bằng	5 != 5 // cho 0
<	So sánh hỏ hơn	5 < 5.5 // cho 1
<=	So sánh hỏ hơn hoặc bằng	5 <= 5 // cho 1
>	So sánh lớn hơn	5 > 5.5 // cho 0
>=	So sánh lớn hơn hoặc bằng	6.3 >= 5 // cho 1

Chú ý rằng các toán tử <= và >= chỉ được hỗ trợ trong hình thức hiển thị. Nói riêng cả hai =< và => đều không hợp lệ và không mang ý nghĩa gì cả.

Các toán hạng của một toán tử quan hệ phải ước lượng về một số. Các ký tự là các toán hạng hợp lệ vì chúng được đại diện bởi các giá trị số. Ví dụ (giả sử mã ASCII):

```
'A' < 'F'            // được 1 (giống như là 65 < 70)
```

Các toán tử quan hệ không nên được dùng để so sánh chuỗi bởi vì điều này sẽ dẫn đến các *địa chỉ* của chuỗi được so sánh chứ không phải là nội dung chuỗi. Ví dụ, biểu thức

```
"HELLO" < "BYE"
```

làm cho địa chỉ của chuỗi "HELLO" được so sánh với địa chỉ của chuỗi "BYE". Vì các địa chỉ này được xác định bởi trình biên dịch, kết quả có thể là 0 hoặc có thể là 1, cho nên chúng ta có thể nói kết quả là không được định nghĩa.

C++ cung cấp các thư viện hàm (ví dụ, strcmp) để thực hiện so sánh chuỗi.

2.3. Toán tử luận lý

C++ cung cấp ba toán tử luận lý cho việc kết nối các biểu thức luận lý. Các toán tử này được tổng kết trong Bảng 2.3. Giống như các toán tử quan hệ, các toán tử luận lý ước lượng tới 0 hoặc 1.

Bảng 2.3 Các toán tử luận lý.

Toán tử	Tên	Ví dụ
!	Phủ định luận lý	!(5 == 5) // được 0
&&	Và luận lý	5 < 6 && 6 < 6 // được 0
||	Hoặc luận lý	5 < 6 || 6 < 5 // được 1

Phủ định luận lý là một toán tử đơn hạng chỉ phủ định giá trị luận lý toán hạng đơn của nó. Nếu toán hạng của nó không là 0 thì được 0, và nếu nó là không thì được 1.

Và luận lý cho kết quả 0 nếu một hay cả hai toán hạng của nó ước lượng tới 0. Ngược lại, nó cho kết quả 1. Hoặc luận lý cho kết quả 0 nếu cả hai toán hạng của nó ước lượng tới 0. Ngược lại, nó cho kết quả 1.

Chú ý rằng ở đây chúng ta nói các toán hạng là 0 và khác 0. Nói chung, bất kỳ giá trị không là 0 nào có thể được dùng để đại diện cho đúng (true), trong khi chỉ có giá trị 0 là đại diện cho sai (false). Tuy nhiên, tất cả các hàng sau đây là các biểu thức luận lý hợp lệ:

```
!20                    // được 0
10 && 5                // được 1
10 || 5.5              // được 1
10 && 0                // được 0
```

C++ không có kiểu boolean xây dựng sẵn. Vì lẽ đó mà ta có thể sử dụng kiểu int cho mục đích này. Ví dụ:

```
int    sorted = 0;     // false
int    balanced = 1;   // true
```

2.4. Toán tử trên bit

C++ cung cấp 6 toán tử trên bit để điều khiển các bit riêng lẻ trong một số lượng số nguyên. Chúng được tổng kết trong Bảng 2.4.

Bảng 2.4 Các toán tử trên bit.

Toán tử	Tên	Ví dụ
~	Phủ định bit	~'\011' // được '\366'
&	Và bit	'\011' & '\027' // được '\001'
|	Hoặc bit	'\011' | '\027' // được '\037'
^	Hoặc exclusive bit	'\011' ^ '\027' // được '\036'
<<	Dịch trái bit	'\011' << 2 // được '\044'
>>	Dịch phải bit	'\011' >> 2 // được '\002'

Các toán tử trên bit mong đợi các toán hạng của chúng là các số nguyên và xem chúng như là một chuỗi các bit. *Phủ định bit* là một toán tử đơn hạng thực hiện đảo các bit trong toán hạng của nó. *Và bit* so sánh các bit tương ứng của các toán hạng của nó và cho kết quả là 1 khi cả hai bit là 1, ngược lại là 0. *Hoặc bit* so sánh các bit tương ứng của các toán hạng của nó và cho kết quả là 0 khi cả hai bit là 0, ngược lại là 1. *XOR bit* so sánh các bit tương ứng của các toán hạng của nó và cho kết quả 0 khi cả hai bit là 1 hoặc cả hai bit là 0, ngược lại là 1.

Cả hai toán tử *dịch trái bit* và *dịch phải bit* lấy một chuỗi bit làm toán hạng trái của chúng và một số nguyên dương *n* làm toán hạng phải. Toán tử dịch trái cho kết quả là một chuỗi bit sau khi thực hiện dịch *n* bit trong chuỗi bit của toán hạng trái về phía trái. Toán tử dịch phải cho kết quả là một chuỗi bit sau khi thực hiện dịch *n* bit trong chuỗi bit của toán hạng trái về phía phải. Các bit trống sau khi dịch được đặt tới 0.

Bảng 2.5 minh họa chuỗi các bit cho các toán hạng ví dụ và kết quả trong Bảng 2.4. Để tránh lo lắng về bit dấu (điều này phụ thuộc vào máy) thường thì khai báo chuỗi bit như là một số không dấu:

```
unsigned char x = '\011';
unsigned char y = '\027';
```

Bảng 2.5 Các bit được tính toán như thế nào.

Ví dụ	Giá trị cơ số 8	Chuỗi bit							
x	011	0	0	0	0	1	0	0	1
y	027	0	0	0	1	0	1	1	1
~x	366	1	1	1	1	0	1	1	0
x&y	001	0	0	0	0	0	0	0	1
x|y	037	0	0	0	1	1	1	1	1
x^y	036	0	0	0	1	1	1	1	0
x<<2	044	0	0	1	0	0	1	0	0
x>>2	002	0	0	0	0	0	0	1	0

2.5. Toán tử tăng/giảm

Các toán tử *tăng một* (++) và *giảm một* (--) cung cấp các tiện lợi tương ứng cho việc cộng thêm 1 vào một biến số hay trừ đi 1 từ một biến số. Các toán tử này được tổng kết trong Bảng 2.6. Các ví dụ giả sử đã định nghĩa biến sau:

```
int    k=5;
```

Bảng 2.6 Các toán tử tăng và giảm.

Toán tử	Tên	Ví dụ	
++	Tăng một (tiền tố)	++k+10	// được 16
++	Tăng một (hậu tố)	k+++10	// được 15
--	Giảm một (tiền tố)	--k+10	// được 14
--	Giảm một (hậu tố)	k--+10	// được 15

Cả hai toán tử có thể được sử dụng theo hình thức tiền tố hay hậu tố là hoàn toàn khác nhau. Khi được sử dụng theo hình thức tiền tố thì toán tử được áp dụng trước và kết quả sau đó được sử dụng trong biểu thức. Khi được sử dụng theo hình thức hậu tố thì biểu thức được ước lượng trước và sau đó toán tử được áp dụng.

Cả hai toán tử có thể được áp dụng cho biến nguyên cũng như là biến thực mặc dù trong thực tế thì các biến thực hiếm khi được dùng theo hình thức này.

2.6. Toán tử khởi tạo

Toán tử khởi tạo được sử dụng để lưu trữ một biến. Toán hạng trái nên là một giá trị trái và toán hạng phải có thể là một biểu thức bất kỳ. Biểu thức được ước lượng và kết quả được lưu trữ trong vị trí được chỉ định bởi giá trị trái.

Giá trị trái là bất kỳ thứ gì chỉ định rõ vị trí bộ nhớ lưu trữ một giá trị. Chỉ một loại của giá trị trái mà chúng ta được biết cho đến thời điểm này là

biến. Các loại khác của giá trị trái (dựa trên con trỏ và tham chiếu) sẽ được thảo luận sau.

Toán tử khởi tạo có một số biến thể thu được bằng cách kết nối nó với các toán tử toán học và các toán tử trên bit. Chúng được tổng kết trong Bảng 2.7. Các ví dụ giả sử rằng n là một biến số nguyên.

Bảng 2.7 Các toán tử khởi tạo.

Toán tử	Ví dụ	Tương đương với
=	n=25	
+=	n+=25	n=n+25
-=	n-=25	n=n-25
*=	n*=25	n=n*25
/=	n/=25	n=n/25
%=	n%=25	n=n%25
&=	n&=0xF2F2	n=n&0xF2F2
|=	n|=0xF2F2	n=n|0xF2F2
^=	n^=0xF2F2	n=n^0xF2F2
<<=	n<<=4	n=n<<4
>>=	n>>=4	n=n>>4

Phép toán khởi tạo chính nó là một biểu thức mà giá trị của nó là giá trị được lưu trong toán hạng trái của nó. Vì thế một phép toán khởi tạo có thể được sử dụng như là toán hạng phải của một phép toán khởi tạo khác. Bất kỳ số lượng khởi tạo nào có thể được kết nối theo cách này để hình thành một biểu thức. Ví dụ:

```
int m, n, p;
m=n=p=100;              // nghĩa là: n=(m=(p=100));
m=(n=p=100)+2; // nghĩa là: m=(n=(p=100))+2;
```

Việc này có thể ứng dụng tương tự cho các hình thức khởi tạo khác. Ví dụ:

```
m=100;
m+=n=p=10;              // nghĩa là: m=m+(n=p=10);
```

2.7. Toán tử điều kiện

Toán tử điều kiện yêu cầu 3 toán hạng. Hình thức chung của nó là:

toán hạng 1 ? *toán hạng 2* : *toán hạng 3*

Toán hạng đầu tiên được ước lượng và được xem như là một điều kiện. Nếu kết quả không là 0 thì toán hạng 2 được ước lượng và giá trị của nó là kết quả sau cùng. Ngược lại, toán hạng 3 được ước lượng và giá trị của nó là kết quả sau cùng. Ví dụ:

```
int m = 1, n = 2;
int min = (m < n ? m : n);          // min nhận giá trị 1
```

Chú ý rằng trong các toán hạng thứ 2 và toán hạng thứ 3 của toán tử điều kiện thì chỉ có một toán hạng được thực hiện. Điều này là quan trọng khi một hoặc cả hai chứa hiệu ứng phụ (nghĩa là, việc ước lượng của chúng làm chuyển đổi giá trị của biến). Ví dụ, với m=1 và n=2 thì trong

```
int min = (m < n ? m++ : n++);
```

m được tăng lên bởi vì m++ được ước lượng nhưng n không tăng vì n++ không được ước lượng.

Bởi vì chính phép toán điều kiện cũng là một biểu thức nên nó có thể được sử dụng như một toán hạng của phép toán điều kiện khác, có nghĩa là các biểu thức điều kiện có thể được lồng nhau. Ví dụ:

```
int m = 1, n = 2, p=3;
int min = (m < n  ? (m < p ? m : p)
                  : (n < p ? n : p));
```

2.8. Toán tử phẩy

Nhiều biểu thức có thể được kết nối vào cùng một biểu thức sử dụng toán tử phẩy. Toán tử phẩy yêu cầu 2 toán hạng. Đầu tiên nó ước lượng toán hạng trái sau đó là toán hạng phải, và trả về giá trị của toán hạng phải như là kết quả sau cùng. Ví dụ:

```
int m=1, n=2, min;
int mCount = 0, nCount = 0;
//...
min = (m < n ? mCount++, m : nCount++, n);
```

Ở đây khi m nhỏ hơn n, mCount++ được ước lượng và giá trị của m được lưu trong min. Ngược lại, nCount++ được ước lượng và giá trị của n được lưu trong min.

2.9. Toán tử lấy kích thước

C++ cung cấp toán tử hữu dụng, sizeof, để tính toán kích thước của bất kỳ hạng mục dữ liệu hay kiểu dữ liệu nào. Nó yêu cầu một toán hạng duy nhất có thể là tên kiểu (ví dụ, int) hay một biểu thức (ví dụ, 100) và trả về kích thước của những thực thể đã chỉ định theo byte. Kết quả hoàn toàn phụ thuộc vào máy. Danh sách 2.1 minh họa việc sử dụng toán tử sizeof cho các kiểu có sẵn mà chúng ta đã gặp cho đến thời điểm này.

Danh sách 2.1

```
1   #include <iostream.h>

2   int main (void)
3   {
4         cout << "char  size = " << sizeof(char) << " bytes\n";
5         cout << "char* size = " << sizeof(char*) << " bytes\n";
6         cout << "short size = " << sizeof(short) << " bytes\n";
7         cout << "int   size = " << sizeof(int) << " bytes\n";
8         cout << "long  size = " << sizeof(long) << " bytes\n";
9         cout << "float size = " << sizeof(float) << " bytes\n";
10        cout << "double size = " << sizeof(double) << " bytes\n";

11      cout << "1.55  size = " << sizeof(1.55) << " bytes\n";
12        cout << "1.55L size = " << sizeof(1.55L) << " bytes\n";
13        cout << "HELLO size = " << sizeof("HELLO") << " bytes\n";
14  }
```

Khi chạy, chương trình sẽ cho kết quả sau (trên máy tính cá nhân):

```
char  size = 1 bytes
char* size = 2 bytes
short size = 2 bytes
int   size = 2 bytes
long  size = 4 bytes
float size = 4 bytes
double size = 8 bytes
1.55  size = 8 bytes
1.55L size = 10 bytes
HELLO size = 6 bytes
```

2.10.Độ ưu tiên của các toán tử

Thứ tự mà các toán tử được ước lượng trong một biểu thức là rất quan trọng và được xác định theo các luật ưu tiên. Các luật này chia các toán tử C++ ra thành một số mức độ ưu tiên (xem Bảng 2.8). Các toán tử ở mức cao hơn sẽ có độ ưu tiên cao hơn các toán tử có độ ưu tiên thấp hơn.

Bảng 2.8 Độ ưu tiên của các toán tử.

Mức	Toán tử						Loại	Thứ tự
Cao nhất	::						Đơn hạng	Cả hai
	()	[]	->	.			Nhị hạng	Trái tới phải
	+ -	++ --	! ~	* &	new delete	sizeof ()	Đơn hạng	Phải tới trái
	->*	.*					Nhị hạng	Trái tới phải
	*	/	%				Nhị hạng	Trái tới phải
	+	-					Nhị hạng	Trái tới phải
	<<	>>					Nhị hạng	Trái tới phải
	<	<=	>	>=			Nhị hạng	Trái tới phải
	==	!=					Nhị hạng	Trái tới phải
	&						Nhị hạng	Trái tới phải

	^						Nhị hạng	Trái tới phải
	|						Nhị hạng	Trái tới phải
	&&						Nhị hạng	Trái tới phải
	||						Nhị hạng	Trái tới phải
	? :						Tam hạng	Trái tới phải
	=	+= -=	*= /=	^= %=	&= |=	<<= >>=	Nhị hạng	Phải tới trái
Thấp nhất	,						Nhị hạng	Trái tới phải

Ví dụ, trong biểu thức

```
a = b + c * d
```

c * d được ước lượng trước bởi vì toán tử * có độ ưu tiên cao hơn toán tử + và ==. Sau đó kết quả được cộng tới b bởi vì toán tử + có độ ưu tiên cao hơn toán tử ==, và sau đó == được ước lượng. Các luật ưu tiên có thể được cho quyền cao hơn thông qua việc sử dụng các dấu ngoặc. Ví dụ, viết lại biểu thức trên như sau

```
a = (b + c) * d
```

sẽ làm cho toán tử + được ước lượng trước toán tử *.

Các toán tử với cùng mức độ ưu tiên được ước lượng theo thứ tự được ước lượng trong cột cuối cùng trong Bảng 2.8. Ví dụ, trong biểu thức

```
a = b += c
```

thứ tự ước lượng là từ phải sang trái, vì thế b += c được ước lượng trước và kế đó là a = b.

2.11.Chuyển kiểu đơn giản

Một giá trị thuộc về những kiểu xây dựng sẵn mà chúng ta biết đến thời điểm này đều có thể được chuyển về bất kỳ một kiểu nào khác. Ví dụ:

```
(int) 3.14          // chuyển 3.14 sang int để được 3
(long) 3.14         // chuyển 3.14 sang long để được 3L
(double) 2          // chuyển 2 sang double để được 2.0
(char) 122          // chuyển 122 sang char có mã là 122
(unsigned short) 3.14   // được 3 như là một unsigned short
```

Như đã được trình bày trong các ví dụ, các định danh kiểu xây dựng sẵn có thể được sử dụng như **các toán tử kiểu**. Các toán tử kiểu là đơn hạng (nghĩa là chỉ có một toán hạng) và xuất hiện bên trong các dấu ngoặc về bên trái toán hạng của chúng. Điều này được gọi là **chuyển kiểu tường minh.** Khi tên kiểu chỉ là một từ thì có thể đặt dấu ngoặc xung quanh toán hạng:

```
int(3.14)           // như là: (int) 3.14
```

Trong một vài trường hợp, C++ cũng thực hiện **chuyển kiểu không tường minh**. Điều này xảy ra khi các giá trị của các kiểu khác nhau được trộn lẫn trong một biểu thức. Ví dụ:

```
double  d=1;            // d nhận 1.0
int     i=10.5;         // i nhận 10
i=i+d;                  // nghĩa là: i=int(double(i)+d)
```

Trong ví dụ cuối , i + d bao hàm các kiểu không hợp nhau, vì thế trước tiên i được chuyển thành double (*thăng cấp*) và sau đó được cộng vào d. Kết quả là double không hợp kiểu với i trên phía trái của phép gán, vì thế nó được chuyển thành int (*hạ cấp*) trước khi được gán cho i.

Luật trên đại diện cho một vài trường hợp chung đơn giản để chuyển kiểu. Các trường hợp phức tạp hơn sẽ được trình bày ở phần sau của giáo trình sau khi chúng ta thảo luận các kiểu dữ liệu khác.

Bài tập cuối chương 2

2.1 Viết các biểu thức sau đây:

- Kiểm tra một số *n* là chẵn hay không.
- Kiểm tra một ký tự *c* là một số hay không.
- Kiểm tra một ký tự *c* là một mẫu tự hay không.
- Thực hiện kiểm tra: *n* là lẽ và dương hoặc *n* chẵn và âm.
- Đặt lại k bit của một số nguyên *n* tới 0.
- Đặt k bit của một số nguyên *n* tới 1.
- Cho giá trị tuyệt đối của một số *n*.
- Cho số ký tự trong một chuỗi *s* được kết thúc bởi ký tự null.

2.2 Thêm các dấu ngoặc phụ vào các biểu thức sau để hiển thị rõ ràng thứ tự các toán tử được ước lượng:

```
(n <= p + q && n >= p - q || n == 0)
        (++n*q--/++p-q)
        (n|p&q^p<<2+q)
        (p<q?n<p?q*n-2:q/n+1:q-n)
```

2.3 Cho biết giá trị của mỗi biến sau đây sau khi khởi tạo nó:

```
double  d=2*int(3.14);
long    k=3.14-3;
char    c='a'+2;
char    c='p'+'A'-'a';
```

2.4 Viết một chương trình cho phép nhập vào một số nguyên dương *n* và xuất ra giá trị của *n* mũ 2 và 2 mũ *n*.

2.5 Viết một chương trình cho phép nhập ba số và xuất ra thông điệp Sorted nếu các số là tăng dần và xuất ra Not sorted trong trường hợp ngược lại.