ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
Khoa Công nghệ Thông tin
PHẠM HỒNG THÁI

Bài giảng
NGÔN NGỮ LẬP TRÌNH C/C++
(tiếp theo)

IV. PHÉP TOÁN, BIỂU THỨC VÀ CÂU LỆNH

1. Phép toán:

C++ có rất nhiều phép toán loại 1 ngôi, 2 ngôi và thậm chí cả 3 ngôi. Để hệ thống, chúng tôi tạm phân chia thành các lớp và trình bày chỉ một số trong chúng. Các phép toán còn lại sẽ được tìm hiểu dần trong các phần sau của giáo trình. Các thành phần tên gọi tham gia trong phép toán được gọi là hạng thức hoặc toán hạng, các kí hiệu phép toán được gọi là toán tử. Ví dụ trong phép toán a + b; a, b được gọi là toán hạng và + là toán tử. Phép toán 1 ngôi là phép toán chỉ có một toán hạng, ví dụ −a (đổi dấu số a), &x (lấy địa chỉ của biến x) … Một số kí hiệu phép toán cũng được sử dụng chung cho cả 1 ngôi lẫn 2 ngôi (hiển nhiên với ngữ nghĩa khác nhau), ví dụ kí hiệu − được sử dụng cho phép toán trừ 2 ngôi a − b, hoặc phép & còn được sử dụng cho phép toán lấy hội các bit (a & b) của 2 số nguyên a và b …

a. Các phép toán số học: +, -, *, /, %

− Các phép toán + (cộng), − (trừ), * (nhân) được hiểu theo nghĩa thông thường trong số học.

− Phép toán a / b (chia) được thực hiện theo kiểu của các toán hạng, tức nếu cả hai toán hạng là số nguyên thì kết quả của phép chia chỉ lấy phần nguyên, ngược lại nếu 1 trong 2 toán hạng là thực thì kết quả là số thực. Ví dụ:

```
13/5 = 2                                  // do 13 và 5 là 2 số nguyên
13.0/5 = 13/5.0 = 13.0/5.0 = 2.6          // do có ít nhất 1 toán hạng là thực
```

– Phép toán a % b (lấy phần dư) trả lại phần dư của phép chia a/b, trong đó a và b là 2 số nguyên. Ví dụ:

```
13%5 = 3                    // phần dư của 13/5
5%13 = 5                    // phần dư của 5/13
```

b. Các phép toán tự tăng, giảm: i++, ++i, i--, --i

– Phép toán ++i và i++ sẽ cùng tăng i lên 1 đơn vị tức tương đương với câu lệnh i = i+1. Tuy nhiên nếu 2 phép toán này nằm trong câu lệnh hoặc biểu thức thì ++i khác với i++. Cụ thể ++i sẽ tăng i, sau đó i mới được tham gia vào tính toán trong biểu thức. Ngược lại i++ sẽ tăng i sau khi biểu thức được tính toán xong (với giá trị i cũ). Điểm khác biệt này được minh hoạ thông qua ví dụ sau, giả sử i = 3, j = 15.

Phép toán	Tương đương	Kết quả
i = ++j ; // tăng trước	j = j + 1 ; i = j ;	i = 16 , j = 16
i = j++ ; // tăng sau	i = j ; j = j + 1 ;	i = 15 , j = 16
j = ++i + 5 ;	i = i + 1 ; j = i + 5 ;	i = 4, j = 9
j = i++ + 5 ;	j = i + 5; i = i + 1;	i = 4, j = 8

Ghi chú: Việc kết hợp phép toán tự tăng giảm vào trong biểu thức hoặc câu lệnh (như ví dụ trong phần sau) sẽ làm chương trình gọn nhưng khó hiểu hơn.

c. Các phép toán so sánh và lôgic

Đây là các phép toán mà giá trị trả lại là đúng hoặc sai. Nếu giá trị của biểu thức là đúng thì nó nhận giá trị 1, ngược lại là sai thì biểu thức nhận giá trị 0. Nói cách khác 1 và 0 là giá trị cụ thể của 2 khái niệm "đúng", "sai". Mở rộng hơn C++ quan niệm một giá trị bất kỳ khác 0 là "đúng" và giá trị 0 là "sai".

- Các phép toán so sánh

  == (bằng nhau), != (khác nhau), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng).

Hai toán hạng của các phép toán này phải cùng kiểu. Ví dụ:

```
3 == 3 hoặc 3 == (4 -1)     // nhận giá trị 1 vì đúng
3 == 5                      // = 0 vì sai
3 != 5                      // = 1
3 + (5 < 2)                 // = 3 vì 5<2 bằng 0
```

```
3 + (5 >= 2)                    // = 4 vì 5>=2 bằng 1
```

Chú ý: cần phân biệt phép toán gán (=) và phép toán so sánh (==). Phép gán vừa gán giá trị cho biến vừa trả lại giá trị bất kỳ (là giá trị của toán hạng bên phải), trong khi phép so sánh luôn luôn trả lại giá trị 1 hoặc 0.

- Các phép toán lôgic:

  && (và), || (hoặc), ! (không, phủ định)

Hai toán hạng của loại phép toán này phải có kiểu lôgic tức chỉ nhận một trong hai giá trị "đúng" (được thể hiện bởi các số nguyên khác 0) hoặc "sai" (thể hiện bởi 0). Khi đó giá trị trả lại của phép toán là 1 hoặc 0 và được cho trong bảng sau:

a	b	a && b	a || b	! a
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Tóm lại:

– Phép toán "và" đúng khi và chỉ khi hai toán hạng cùng đúng

– Phép toán "hoặc" sai khi và chỉ khi hai toán hạng cùng sai

– Phép toán "không" (hoặc "phủ định") đúng khi và chỉ khi toán hạng của nó sai.

Ví dụ:

```
3 && (4 > 5)            // = 0 vì có hạng thức (4>5) sai
(3 >= 1) && (7)         // = 1 vì cả hai hạng thức cùng đúng
!1                      // = 0
! (4 + 3 < 7)           // = 1 vì (4+3<7) bằng 0
5 || (4 >= 6)           // = 1 vì có một hạng thức (5) đúng
(5 < !0) || (4 >= 6)    // = 0 vì cả hai hạng thức đều sai
```

Chú ý: việc đánh giá biểu thức được tiến hành từ trái sang phải và sẽ dừng khi biết kết quả mà không chờ đánh giá hết biểu thức. Cách đánh giá này sẽ cho những kết quả phụ khác nhau nếu trong biểu thức ta "tranh thủ" đưa thêm vào các phép toán tự tăng giảm. Ví dụ cho i = 2, j = 3, xét 2 biểu thức sau đây:

```
x = (++i < 4 && ++j > 5)        cho kết quả x = 0 , i = 3 , j = 4
```

```
y = (++j > 5 && ++i < 4)          cho kết quả y = 0 , i = 2 , j = 4
```

cách viết hai biểu thức là như nhau (ngoại trừ hoán đổi vị trí 2 toán hạng của phép toán &&). Với giả thiết i = 2 và j = 3 ta thấy cả hai biểu thức trên cùng nhận giá trị 0. Tuy nhiên các giá trị của i và j sau khi thực hiện xong hai biểu thức này sẽ có kết quả khác nhau. Cụ thể với biểu thức đầu vì ++i < 4 là đúng nên chương trình phải tiếp tục tính tiếp ++j > 5 để đánh giá được biểu thức. Do vậy sau khi đánh giá xong cả i và j đều được tăng 1 (i=3, j=4). Trong khi đó với biểu thức sau do ++j > 5 là sai nên chương trình có thể kết luận được toàn bộ biểu thức là sai mà không cần tính tiếp ++i < 4. Có nghĩa chương trình sau khi đánh giá xong ++j > 5 sẽ dừng và vì vậy chỉ có biến j được tăng 1, từ đó ta có i = 2, j = 4 khác với kết quả của biểu thức trên. Ví dụ này một lần nữa nhắc ta chú ý kiểm soát kỹ việc sử dụng các phép toán tự tăng giảm trong biểu thức và trong câu lệnh.

2. Các phép gán

- Phép gán thông thường: Đây là phép gán đã được trình bày trong mục trước.
- Phép gán có điều kiện:

biến = (điều_kiện) ? a: b ;

điều_kiện là một biểu thức logic, a, b là các biểu thức bất kỳ cùng kiểu với kiểu của biến. Phép toán này gán giá trị a cho biến nếu điều kiện đúng và b nếu ngược lại.

Ví dụ:

```
x = (3 + 4 < 7) ? 10: 20      // x = 20 vì 3+4<7 là sai
x = (3 + 4) ? 10: 20          // x = 10 vì 3+4 khác 0, tức điều kiện đúng
x = (a > b) ? a: b            // x = số lớn nhất trong 2 số a, b.
```

- Cách viết gọn của phép gán: Một phép gán dạng x = x @ a ; có thể được viết gọn dưới dạng x @= a trong đó @ là các phép toán số học, xử lý bit ... Ví dụ:

  thay cho viết x = x + 2 có thể viết x += 2;

  hoặc x = x/2 ; x = x*2 có thể được viết lại như x /= 2; x *= 2;

Cách viết gọn này có nhiều thuận lợi khi viết và đọc chương trình nhất là khi tên biến quá dài hoặc đi kèm nhiều chỉ số … thay vì phải viết hai lần tên biến trong câu lệnh thì chỉ phải viết một lần, điều này tránh viết lặp lại tên biến dễ gây ra sai sót. Ví dụ thay vì viết:

```
ngay_quoc_te_lao_dong = ngay_quoc_te_lao_dong + 365;
```

có thể viết gọn hơn bởi:

```
ngay_quoc_te_lao_dong += 365;
```

hoặc thay cho viết :

```
Luong[Nhanvien[3][2*i+1]] = Luong[Nhanvien[3][2*i+1]] * 290 ;
```

có thể được viết lại bởi:

```
Luong[Nhanvien[3][2*i+1]] *= 290;
```

3. Biểu thức

Biểu thức là dãy kí hiệu kết hợp giữa các toán hạng, phép toán và cặp dấu () theo một qui tắc nhất định. Các toán hạng là hằng, biến, hàm. Biểu thức cung cấp một cách thức để tính giá trị mới dựa trên các toán hạng và toán tử trong biểu thức. Ví dụ:

```
(x + y) * 2 - 4 ;       3 - x + sqrt(y) ;  (-b + sqrt(delta)) / (2*a) ;
```

a. Thứ tự ưu tiên của các phép toán

Để tính giá trị của một biểu thức cần có một trật tự tính toán cụ thể và thống nhất. Ví dụ xét biểu thức x = 3 + 4 * 2 + 7

- nếu tính theo đúng trật tự từ trái sang phải, ta có x = ((3+4) * 2) + 7 = 21,
- nếu ưu tiên dấu + được thực hiện trước dấu *, x = (3 + 4) * (2 + 7) = 63,
- nếu ưu tiên dấu * được thực hiện trước dấu +, x = 3 + (4 * 2) + 7 = 18.

Như vậy cùng một biểu thức tính x nhưng cho 3 kết quả khác nhau theo những cách hiểu khác nhau. Vì vậy cần có một cách hiểu thống nhất dựa trên thứ tự ưu tiên của các phép toán, tức những phép toán nào sẽ được ưu tiên tính trước và những phép toán nào được tính sau ...

C++ qui định trật tự tính toán theo các mức độ ưu tiên như sau:

1. Các biểu thức trong cặp dấu ngoặc ()
2. Các phép toán 1 ngôi (tự tăng, giảm, lấy địa chỉ, lấy nội dung con trỏ …)
3. Các phép toán số học.
4. Các phép toán quan hệ, logic.
5. Các phép gán.

Nếu có nhiều cặp ngoặc lồng nhau thì cặp trong cùng (sâu nhất) được tính trước. Các phép toán trong cùng một lớp có độ ưu tiên theo thứ tự: lớp nhân (*, /, &&), lớp cộng (+, –, ||). Nếu các phép toán có cùng thứ tự ưu tiên thì chương trình sẽ thực hiện từ trái sang phải. Các phép gán có độ ưu tiên cuối cùng và được thực hiện từ phải sang trái. Ví dụ. theo mức ưu tiên đã qui định, biểu thức tính x trong ví dụ trên sẽ được tính như x = 3 + (4 * 2) + 7 = 18.

Phần lớn các trường hợp muốn tính toán theo một trật tự nào đó ta nên sử dụng cụ thể các dấu ngoặc (vì các biểu thức trong dấu ngoặc được tính trước). Ví dụ:

- Để tính $\Delta = b^2 - 4ac$ ta viết delta = b * b – 4 * a * c ;
- Để tính nghiệm phương trình bậc 2: $x = \frac{-b+\sqrt{\Delta}}{2a}$ viết : x = –b + sqrt(delta) / 2*a; là sai vì theo mức độ ưu tiên x sẽ được tính như –b + ((sqrt(delta)/2) * a) (thứ tự tính sẽ là phép toán 1 ngôi đổi dấu –b, đến phép chia, phép nhân và cuối cùng là phép cộng). Để tính chính xác cần phải viết (–b + sqrt(delta)) / (2*a).
- Cho a = 1, b = 2, c = 3. Biểu thức a += b += c cho giá trị c = 3, b = 5, a = 6. Thứ tự tính sẽ là từ phải sang trái, tức câu lệnh trên tương đương với các câu lệnh sau:

```
a = 1 ; b = 2 ; c = 3 ;
b = b + c ;                    // b = 5
a = a + b ;                    // a = 6
```

Để rõ ràng, tốt nhất nên viết biểu thức cần tính trước trong các dấu ngoặc.

b. Phép chuyển đổi kiểu

Khi tính toán một biểu thức phần lớn các phép toán đều yêu cầu các toán hạng phải cùng kiểu. Ví dụ để phép gán thực hiện được thì giá trị của biểu thức phải có **cùng kiểu** với biến. Trong trường hợp kiểu của giá trị biểu thức khác với kiểu của phép gán thì hoặc là chương trình sẽ tự động chuyển kiểu giá trị biểu thức về thành kiểu của biến được gán (nếu được) hoặc sẽ báo lỗi. Do vậy khi cần thiết NSD phải sử dụng các câu lệnh để chuyển kiểu của biểu thức cho phù hợp với kiểu của biến.

- Chuyển kiểu tự động: về mặt nguyên tắc, khi cần thiết các kiểu có giá trị thấp sẽ được chương trình tự động chuyển lên kiểu cao hơn cho phù hợp với phép toán. Cụ thể phép chuyển kiểu có thể được thực hiện theo sơ đồ như sau:

  char $\leftrightarrow$ int $\rightarrow$ long int $\rightarrow$ float $\rightarrow$ double

Ví dụ:

```
int i = 3;
float f ;
f = i + 2;
```

trong ví dụ trên i có kiểu nguyên và vì vậy i+2 cũng có kiểu nguyên trong khi f có kiểu thực. Tuy vậy phép toán gán này là hợp lệ vì chương trình sẽ tự động chuyển kiểu

cuả i+2 (bằng 5) sang kiểu thực (bằng 5.0) rồi mới gán cho f.

– Ép kiểu: trong chuyển kiểu tự động, chương trình chuyển các kiểu từ thấp đến cao, tuy nhiên chiều ngược lại không thể thực hiện được vì nó có thể gây mất dữ liệu. Do đó nếu cần thiết NSD phải ra lệnh cho chương trình. Ví dụ:

```
int i;
float f = 3 ;          // tự động chuyển 3 thành 3.0 và gán cho f
i = f + 2 ;            // sai vì mặc dù f + 2 = 5 nhưng không gán được cho i
```

Trong ví dụ trên để câu lệnh i = f+2 thực hiện được ta phải ép kiểu của biểu thức f+2 về thành kiểu nguyên. Cú pháp tổng quát như sau:

```
(tên_kiểu)biểu_thức          // cú pháp cũ trong C
```

hoặc:

```
tên_kiểu(biểu_thức)          // cú pháp mới trong C++
```

trong đó tên_kiểu là kiểu cần được chuyển sang. Như vậy câu lệnh trên phải được viết lại:

```
i = int(f + 2) ;
```

khi đó f+2 (bằng 5.0) được chuyển thành 5 và gán cho i.

Dưới đây ta sẽ xét một số ví dụ về lợi ích của việc ép kiểu.

- Phép ép kiểu từ một số thực về số nguyên sẽ cắt bỏ tất cả phần thập phân của số thực, chỉ để lại phần nguyên. Như vậy để tính phần nguyên của một số thực x ta chỉ cần ép kiểu của x về thành kiểu nguyên, có nghĩa int(x) là phần nguyên của số thực x bất kỳ. Ví dụ để kiểm tra một số nguyên n có phải là số chính phương, ta cần tính căn bậc hai của n. Nếu căn bậc hai x của n là số nguyên thì n là số chính phương, tức nếu int(x) = x thì x nguyên và n là chính phương, ví dụ:

```
int n = 10 ;
float x = sqrt(n) ;          // hàm sqrt(n) trả lại căn bậc hai của số n
if (int(x) == x) cout << "n chính phương" ;
else cout << "n không chính phương" ;
```

- Để biết mã ASCII của một kí tự ta chỉ cần chuyển kí tự đó sang kiểu nguyên.

```
char c ;
cin >> c ;
cout << "Mã của kí tự vừa nhập là " << int(c) ;
```

Ghi chú: Xét ví dụ sau:

```
int i = 3 , j = 5 ;
float x ;
x = i / j * 10;              // x = 6 ?
cout << x ;
```

trong ví dụ này mặc dù x được khai báo là thực nhưng kết quả in ra sẽ là 0 thay vì 6 như mong muốn. Lý do là vì phép chia giữa 2 số nguyên i và j sẽ cho lại số nguyên, tức i/j = 3/5 = 0. Từ đó x = 0*10 = 0. Để phép chia ra kết quả thực ta cần phải ép kiểu hoặc i hoặc j hoặc cả 2 thành số thực, khi đó phép chia sẽ cho kết quả thực và x được tính đúng giá trị. Cụ thể câu lệnh x = i/j*10 được đổi thành:

```
x = float(i) / j * 10 ;              // đúng
x = i / float(j) * 10      ;         // đúng
x = float(i) / float(j) * 10 ;       // đúng
x = float(i/j) * 10 ;                // sai
```

Phép ép kiểu: x = float(i/j) * 10 ; vẫn cho kết quả sai vì trong dấu ngoặc phép chia i/j vẫn là phép chia nguyên, kết quả x vẫn là 0.

4. Câu lệnh và khối lệnh

Một **câu lệnh** trong C++ được thiết lập từ các từ khoá và các biểu thức … và luôn luôn được kết thúc bằng dấu chấm phẩy. Các ví dụ vào/ra hoặc các phép gán tạo thành những câu lệnh đơn giản như:

```
cin >> x >> y ;
x = 3 + x ; y = (x = sqrt(x)) + 1 ;
cout << x ;
cout << y ;
```

Các câu lệnh được phép viết trên cùng một hoặc nhiều dòng. Một số câu lệnh được gọi là lệnh có cấu trúc, tức bên trong nó lại chứa dãy lệnh khác. Dãy lệnh này phải được bao giữa cặp dấu ngoặc {} và được gọi là *khối lệnh*. Ví dụ tất cả các lệnh trong một hàm (như hàm main()) luôn luôn là một khối lệnh. Một đặc điểm của khối lệnh là các biến được khai báo trong khối lệnh nào thì chỉ có tác dụng trong khối lệnh đó. Chi tiết hơn về các đặc điểm của lệnh và khối lệnh sẽ được trình bày trong các chương tiếp theo của giáo trình.

V. THƯ VIỆN CÁC HÀM TOÁN HỌC

Trong phần này chúng tôi tóm tắt một số các hàm toán học hay dùng. Các hàm này đều được khai báo trong file nguyên mẫu math.h.

1. Các hàm số học

- abs(x), labs(x), fabs(x) : trả lại giá trị tuyệt đối của một số nguyên, số nguyên dài và số thực.
- pow(x, y) : hàm mũ, trả lại giá trị x lũy thừa y (x^y).
- exp(x) : hàm mũ, trả lại giá trị e mũ x (e^x).
- log(x), log10(x) : trả lại lôgarit cơ số e và lôgarit thập phân của x (lnx, logx) .
- sqrt(x) : trả lại căn bậc 2 của x.
- atof(s_number) : trả lại số thực ứng với số viết dưới dạng xâu kí tự s_number.

2. Các hàm lượng giác

- sin(x), cos(x), tan(x) : trả lại các giá trị sinx, cosx, tgx.

BÀI TẬP

1. Viết câu lệnh khai báo biến để lưu các giá trị sau:

- Tuổi của một người
- Số lượng cây trong thành phố
- Độ dài cạnh một tam giác
- Khoảng cách giữa các hành tinh
- Một chữ số
- Nghiệm x của phương trình bậc 1
- Một chữ cái
- Biệt thức Δ của phương trình bậc 2

2. Viết câu lệnh nhập vào 4 giá trị lần lượt là số thực, nguyên, nguyên dài và kí tự. In ra màn hình các giá trị này để kiểm tra.

3. Viết câu lệnh in ra màn hình các dòng sau (không kể các số thứ tự và dấu: ở đầu mỗi dòng)

1: Bộ Giáo dục và Đào tạo Cộng hoà xã hội chủ nghĩa Việt Nam

2:

3: Sở Giáo dục Hà Nội Độc lập - Tự do - Hạnh phúc

Chú ý: khoảng trống giữa chữ Đào tạo và Cộng hoà (dòng 1) là 2 tab. Dòng 2: để trống.

4. Viết chương trình nhập vào một kí tự. In ra kí tự đó và mã ascii của nó.

5. Viết chương trình nhập vào hai số thực. In ra hai số thực đó với 2 số lẻ và cách nhau 5 cột.

6. Nhập, chạy và giải thích kết quả đạt được của đoạn chương trình sau:

```
#include <iostream.h>
void main()
{
    char c1 = 200; unsigned char c2 = 200 ;
    cout << "c1 = " << c1 << ", c2 = " << c2 << "\n" ;
    cout << "c1+100 = " << c1+100 << ", c2+100 = " << c2+100 ;
}
```

7. Nhập a, b, c. In ra màn hình dòng chữ phương trình có dạng ax^2 + bx + c = 0, trong đó các giá trị a, b, c chỉ in 2 số lẻ (ví dụ với a = 5.141, b = –2, c = 0.8 in ra 5.14 x^2 –2.00 x + 0.80).

8. Viết chương trình tính và in ra giá trị các biểu thức sau với 2 số lẻ:

a. $\sqrt{3+\sqrt{3+\sqrt{3}}}$ b. $\dfrac{1}{2+\dfrac{1}{2+\dfrac{1}{2}}}$

9. Nhập a, b, c là các số thực. In ra giá trị của các biểu thức sau với 3 số lẻ:

a. $a^2 - 2b + ab/c$ c. $3a - b^3 - 2\sqrt{c}$

b. $\dfrac{b^2-4ac}{2a}$ d. $\sqrt{a^2/b-4a/bc+1}$

10. In ra tổng, tích, hiệu và thương của 2 số được nhập vào từ bàn phím.

11. In ra trung bình cộng, trung bình nhân của 3 số được nhập vào từ bàn phím.

12. Viết chương trình nhập cạnh, bán kính và in ra diện tích, chu vi của các hình: vuông, chữ nhật, tròn.

13. Nhập a, b, c là độ dài 3 cạnh của tam giác (chú ý đảm bảo tổng 2 cạnh phải lớn