CHƯƠNG 2

KIỂU DỮ LIỆU, BIỂU THỨC VÀ CÂU LỆNH

Kiểu dữ liệu đơn giản
Hằng - khai báo và sử dụng hằng
Biến - khai báo và sử dụng biến
Phép toán, biểu thức và câu lệnh
Thư viện các hàm toán học

I. KIỂU DỮ LIỆU ĐƠN GIẢN

1. Khái niệm về kiểu dữ liệu

Thông thường dữ liệu hay dùng là số và chữ. Tuy nhiên việc phân chia chỉ 2 loai dữ liệu là không đủ. Để dễ dàng hơn cho lập trình, hầu hết các NNLT đều phân chia dữ liệu thành nhiều kiểu khác nhau được gọi là các kiểu cơ bản hay chuẩn. Trên cơ sở kết hợp các kiểu dữ liệu chuẩn, NSD có thể tự đặt ra các kiểu dữ liệu mới để phục vụ cho chương trình giải quyết bài toán của mình. Có nghĩa lúc đó mỗi đối tượng được quản lý trong chương trình sẽ là một tập hợp nhiều thông tin hơn và được tạo thành từ nhiều loại (kiểu) dữ liệu khác nhau. Dưới đây chúng ta sẽ xét đến một số kiểu dữ liệu chuẩn được qui định sẵn bởi C++.

Một biến như đã biết là một số ô nhớ liên tiếp nào đó trong bộ nhớ dùng để lưu trữ dữ liệu (vào, ra hay kết quả trung gian) trong quá trình hoạt động của chương trình. Để quản lý chặt chẽ các biến, NSD cần khai báo cho chương trình biết trước tên biến và kiểu của dữ liệu được chứa trong biến. Việc khai báo này sẽ làm chương trình quản lý các biến dễ dàng hơn như trong việc phân bố bộ nhớ cũng như quản lý các tính toán trên biến theo nguyên tắc: chỉ có các dữ liệu cùng kiểu với nhau mới được phép làm toán với nhau. Do đó, khi đề cập đến một kiểu chuẩn của một NNLT, thông thường chúng ta sẽ xét đến các yếu tố sau:

- tên kiểu: là một từ dành riêng để chỉ định kiểu của dữ liệu.
- số byte trong bộ nhớ để lưu trữ một đơn vị dữ liệu thuộc kiểu này: Thông thường số byte này phụ thuộc vào các trình biên dịch và hệ thống máy khác nhau, ở đây ta chỉ xét đến hệ thống máy PC thông dụng hiện nay.
- Miền giá trị của kiểu: Cho biết một đơn vị dữ liệu thuộc kiểu này sẽ có thể lấy

giá trị trong miền nào, ví dụ nhỏ nhất và lớn nhất là bao nhiêu. Hiển nhiên các giá trị này phụ thuộc vào số byte mà hệ thống máy qui định cho từng kiểu. NSD cần nhớ đến miền giá trị này để khai báo kiểu cho các biến cần sử dụng một cách thích hợp.

Dưới đây là bảng tóm tắt một số kiểu chuẩn đơn giản và các thông số của nó được sử dụng trong C++.

Loại dữ liệu	Tên kiểu	Số ô nhớ	Miền giá trị
Kí tự	char	1 byte	– 128 .. 127
	unsigned char	1 byte	0 .. 255
Số nguyên	int	2 byte	– 32768 .. 32767
	unsigned int	2 byte	0 .. 65535
	short	2 byte	– 32768 .. 32767
	long	4 byte	$-2^{15} .. 2^{15} - 1$
Số thực	float	4 byte	$\pm 10^{-37} . . \pm 10^{+38}$
	double	8 byte	$\pm 10^{-307} . . \pm 10^{+308}$

Bảng 1. Các loại kiểu đơn giản

Trong chương này chúng ta chỉ xét các loại kiểu đơn giản trên đây. Các loại kiểu có cấu trúc do người dùng định nghĩa sẽ được trình bày trong các chương sau.

2. Kiểu ký tự

Một kí tự là một kí hiệu trong bảng mã ASCII. Như đã biết một số kí tự có mặt chữ trên bàn phím (ví dụ các chữ cái, chữ số) trong khi một số kí tự lại không (ví dụ kí tự biểu diễn việc lùi lại một ô trong văn bản, kí tự chỉ việc kết thúc một dòng hay kết thúc một văn bản). Do vậy để biểu diễn một kí tự người ta dùng chính mã ASCII của kí tự đó trong bảng mã ASCII và thường gọi là giá trị của kí tự. Ví dụ phát biểu "Cho kí tự 'A'" là cũng tương đương với phát biểu "Cho kí tự 65" (65 là mã ASCII của kí tự 'A'), hoặc "Xoá kí tự xuống dòng" là cũng tương đương với phát biểu "Xoá kí tự 13" vì 13 là mã ASCII của kí tự xuống dòng.

Như vậy một biến kiểu kí tự có thể được nhận giá trị theo 2 cách tương đương - chữ hoặc giá trị số: ví dụ giả sử `c` là một biến kí tự thì câu lệnh gán `c = 'A'` cũng tương đương với câu lệnh gán `c = 65`. Tuy nhiên để sử dụng giá trị số của một kí tự `c` nào đó ta phải yêu cầu đổi `c` sang giá trị số bằng câu lệnh `int(c)`.

Theo bảng trên ta thấy có 2 loại kí tự là `char` với miền giá trị từ -128 đến 127 và

unsigned char (kí tự không dấu) với miền giá trị từ 0 đến 255. Trường hợp một biến được gán giá trị vượt ra ngoài miền giá trị của kiểu thì giá trị của biến sẽ được tính theo mã bù – (256 – c). Ví dụ nếu gán cho char c giá trị 179 (vượt khỏi miền giá trị đã được qui định của char) thì giá trị thực sự được lưu trong máy sẽ là – (256 – 179) = –77.

Ví dụ 1 :

```
char c, d ;                  // c, d được phép gán giá trị từ -128 đến 127
unsigned e ;                 // e được phép gán giá trị từ 0 đến 255
c = 65 ; d = 179 ;           // d có giá trị ngoài miền cho phép
e = 179; f = 330 ;           // f có giá trị ngoài miền cho phép
cout << c << int(c) ;        // in ra chữ cái 'A' và giá trị số 65
cout << d << int(d) ;        // in ra là kí tự '|' và giá trị số -77
cout << e << int(e)          // in ra là kí tự '|' và giá trị số 179
cout << f << int(f)          // in ra là kí tự 'J' và giá trị số 74
```

Chú ý: Qua ví dụ trên ta thấy một biến nếu được gán giá trị ngoài miền cho phép sẽ dẫn đến kết quả không theo suy nghĩ thông thường. Do vậy nên tuân thủ qui tắc chỉ gán giá trị cho biến thuộc miền giá trị mà kiểu của biến đó qui định. Ví dụ nếu muốn sử dụng biến có giá trị từ 128 .. 255 ta nên khai báo biến dưới dạng kí tự không dấu (unsigned char), còn nếu giá trị vượt quá 255 ta nên chuyển sang kiểu nguyên (int) chẳng hạn.

3. Kiểu số nguyên

Các số nguyên được phân chia thành 4 loại kiểu khác nhau với các miền giá trị tương ứng được cho trong bảng 1. Đó là kiểu số nguyên ngắn (short) tương đương với kiểu số nguyên (int) sử dụng 2 byte và số nguyên dài (long int) sử dụng 4 byte. Kiểu số nguyên thường được chia làm 2 loại có dấu (int) và không dấu (unsigned int hoặc có thể viết gọn hơn là unsigned). Qui tắc mã bù cũng được áp dụng nếu giá trị của biến vượt ra ngoài miền giá trị cho phép, vì vậy cần cân nhắc khi khai báo kiểu cho các biến. Ta thường sử dụng kiểu int cho các số nguyên trong các bài toán với miền giá trị vừa phải (có giá trị tuyệt đối bé hơn 32767), chẳng hạn các biến đếm trong các vòng lặp, ...

4. Kiểu số thực

Để sử dụng số thực ta cần khai báo kiểu float hoặc double mà miền giá trị của chúng được cho trong bảng 1. Các giá trị số kiểu double được gọi là số thực với độ chính xác gấp đôi vì với kiểu dữ liệu này máy tính có cách biểu diễn khác so với kiểu

float để đảm bảo số số lẻ sau một số thực có thể tăng lên đảm bảo tính chính xác cao hơn so với số kiểu float. Tuy nhiên, trong các bài toán thông dụng thường ngày độ chính xác của số kiểu float là đủ dùng.

Như đã nhắc đến trong phần các lệnh vào/ra ở chương 1, liên quan đến việc in ấn số thực ta có một vài cách thiết đặt dạng in theo ý muốn, ví dụ độ rộng tối thiểu để in một số hay số số lẻ thập phân cần in ...

Ví dụ 2 : Chương trình sau đây sẽ in diện tích và chu vi của một hình tròn có bán kính 2cm với 3 số lẻ.

```
#include <iostream.h>
#include <iomanip.h>
void main()
{
        float r = 2 ;                    // r là tên biến dùng để chứa bán kính
        cout << "Diện tích = " << setiosflags(ios::showpoint) ;
        cout << setprecision(3) << r * r * 3.1416 ;
        getch() ;
}
```

II. HẰNG - KHAI BÁO VÀ SỬ DỤNG HẰNG

Hằng là một giá trị cố định nào đó ví dụ 3 (hằng nguyên), 'A' (hằng kí tự), 5.0 (hằng thực), "Ha noi" (hằng xâu kí tự). Một giá trị có thể được hiểu dưới nhiều kiểu khác nhau, do vậy khi viết hằng ta cũng cần có dạng viết thích hợp.

1. Hằng nguyên

- kiểu short, int: 3, -7, ...
- kiểu unsigned: 3, 123456, ...
- kiểu long, long int: 3L, -7L, 123456L, ... (viết L vào cuối mỗi giá trị)

Các cách viết trên là thể hiện của số nguyên trong hệ thập phân, ngoài ra chúng còn được viết dưới các hệ đếm khác như hệ cơ số 8 hoặc hệ cơ số 16. Một số nguyên trong cơ số 8 luôn luôn được viết với số 0 ở đầu, tương tự với cơ số 16 phải viết với 0x ở đầu. Ví dụ ta biết 65 trong cơ số 8 là 101 và trong cơ số 16 là 41, do đó 3 cách viết 65, 0101, 0x41 là như nhau, cùng biểu diễn giá trị 65.

2. Hằng thực

Một số thực có thể được khai báo dưới dạng kiểu float hoặc double và các giá trị của nó có thể được viết dưới một trong hai dạng.

a. Dạng dấu phảy tĩnh

Theo cách viết thông thường. Ví dụ: 3.0, -7.0, 3.1416, ...

b. Dạng dấu phảy động

Tổng quát, một số thực x có thể được viết dưới dạng: men hoặc mEn, trong đó m được gọi là phần định trị, n gọi là phần bậc (hay mũ). Số men biểu thị giá trị $x = m \times 10^n$. Ví dụ số $\pi = 3.1416$ có thể được viết:

$\pi = \ldots = 0.031416e2 = 0.31416e1 = 3.1416e0 = 31.416e{-}1 = 314.16e{-}2 = \ldots$

vì $\pi = 0.031416 \times 10^2 = 0.31416 \times 10^1 = 3.1416 \times 10^0 = \ldots$

Như vậy một số x có thể được viết dưới dạng mEn với nhiều giá trị m, n khác nhau, phụ thuộc vào dấu phảy ngăn cách phần nguyên và phần thập phân của số. Do vậy cách viết này được gọi là dạng dấu phảy động.

3. Hằng kí tự

a. Cách viết hằng

Có 2 cách để viết một hằng kí tự. Đối với các kí tự có mặt chữ thể hiện ta thường sử dụng cách viết thông dụng đó là đặt mặt chữ đó giữa 2 dấu nháy đơn như: 'A', '3', ' ' (dấu cách) ... hoặc sử dụng trực tiếp giá trị số của chúng. Ví dụ các giá trị tương ứng của các kí tự trên là 65, 51 và 32. Với một số kí tự không có mặt chữ ta buộc phải dùng giá trị (số) của chúng, như viết 27 thay cho kí tự được nhấn bởi phím Escape, 13 thay cho kí tự được nhấn bởi phím Enter ...

Để biểu diễn kí tự bằng giá trị số ta có thể viết trực tiếp (không dùng cặp dấu nháy đơn) giá trị đó dưới dạng hệ số 10 (như trên) hoặc đặt chúng vào cặp dấu nháy đơn, trường hợp này chỉ dùng cho giá trị viết dưới dạng hệ 8 hoặc hệ 16 theo mẫu sau:

- '\kkk': không quá 3 chữ số trong hệ 8. Ví dụ '\11' biểu diễn kí tự có mã 9.
- '\xkk': không quá 2 chữ số trong hệ 16. Ví dụ '\x1B' biểu diễn kí tự có mã 27.

Tóm lại, một kí tự có thể có nhiều cách viết, chẳng hạn 'A' có giá trị là 65 (hệ 10) hoặc 101 (hệ 8) hoặc 41 (hệ 16), do đó kí tự 'A' có thể viết bởi một trong các dạng sau:

65, 0101, 0x41 hoặc 'A' , '\101' , '\x41'

Tương tự, dấu kết thúc xâu có giá trị 0 nên có thể viết bởi 0 hoặc '\0' hoặc '\x0', trong các cách này cách viết '\0' được dùng thông dụng nhất.

b. Một số hằng thông dụng

Đối với một số hằng kí tự thường dùng nhưng không có mặt chữ tương ứng, hoặc các kí tự được dành riêng với nhiệm vụ khác, khi đó thay vì phải nhớ giá trị của chúng ta có thể viết theo qui ước sau:

'\n'	:	biểu thị kí tự xuống dòng (cũng tương đương với endl)
'\t'	:	kí tự tab
'\a'	:	kí tự chuông (tức thay vì in kí tự, loa sẽ phát ra một tiếng 'bíp')
'\r'	:	xuống dòng
'\f'	:	kéo trang
'\\'	:	dấu \
'\?'	:	dấu chấm hỏi ?
'\''	:	dấu nháy đơn '
'\"'	:	dấu nháy kép "
'\kkk'	:	kí tự có mã là kkk trong hệ 8
'\xkk'	:	kí tự có mã là kk trong hệ 16

Ví dụ:

```
cout << "Hôm nay trời \t nắng \a \a \a \n" ;
```

sẽ in ra màn hình dòng chữ "Hôm nay trời" sau đó bỏ một khoảng cách bằng một tab (khoảng 8 dấu cách) rồi in tiếp chữ "nắng", tiếp theo phát ra 3 tiếng chuông và cuối cùng con trỏ trên màn hình sẽ nhảy xuống đầu dòng mới.

Do dấu cách (phím spacebar) không có mặt chữ, nên trong một số trường hợp để tránh nhầm lẫn chúng tôi qui ước sử dụng kí hiệu <> để biểu diễn dấu cách. Ví dụ trong giáo trình này dấu cách (có giá trị là 32) được viết ' ' (dấu nháy đơn bao một dấu cách) hoặc rõ ràng hơn bằng cách viết theo qui ước <>.

4. Hằng xâu kí tự

Là dãy kí tự bất kỳ đặt giữa cặp dấu nháy kép. Ví dụ: "Lớp K43*", "12A4", "A", "<>", "" là các hằng xâu kí tự, trong đó "" là xâu không chứa kí tự nào, các xâu "<>", "A" chứa 1 kí tự ... Số các kí tự giữa 2 dấu nháy kép được gọi là độ dài của xâu. Ví dụ xâu "" có độ dài 0, xâu "<>" hoặc "A" có độ dài 1 còn xâu "Lớp K43*" có độ dài 8.

Chú ý phân biệt giữa 2 cách viết 'A' và "A", tuy chúng cùng biểu diễn chữ cái A nhưng chương trình sẽ hiểu 'A' là một kí tự còn "A" là một xâu kí tự (do vậy chúng được bố trí khác nhau trong bộ nhớ cũng như cách sử dụng chúng là khác nhau). Tương tự ta không được viết " (2 dấu nháy đơn liền nhau) vì không có khái niệm kí tự

"rỗng". Để chỉ xâu rỗng (không có kí tự nào) ta phải viết "" (2 dấu nháy kép liền nhau).

Tóm lại một giá trị có thể được viết dưới nhiều kiểu dữ liệu khác nhau và do đó cách sử dụng chúng cũng khác nhau. Ví dụ liên quan đến khái niệm 3 đơn vị có thể có các cách viết sau tuy nhiên chúng hoàn toàn khác nhau:

- 3 : số nguyên 3 đơn vị
- 3L : số nguyên dài 3 đơn vị
- 3.0 : số thực 3 đơn vị
- '3' : chữ số 3
- "3" : xâu chứa kí tự duy nhất là 3

5. Khai báo hằng

Một giá trị cố định (hằng) được sử dụng nhiều lần trong chương trình đôi khi sẽ thuận lợi hơn nếu ta đặt cho nó một tên gọi, thao tác này được gọi là khai báo hằng. Ví dụ một chương trình quản lý sinh viên với giả thiết số sinh viên tối đa là 50. Nếu số sinh viên tối đa không thay đổi trong chương trình ta có thể đặt cho nó một tên gọi như SOSV chẳng hạn. Trong suốt chương trình bất kỳ chỗ nào xuất hiện giá trị 50 ta đều có thể thay nó bằng SOSV. Tương tự C++ cũng có những tên hằng được đặt sẵn, được gọi là các hằng chuẩn và NSD có thể sử dụng khi cần thiết. Ví dụ hằng π được đặt sẵn trong C++ với tên gọi M_PI. Việc sử dụng tên hằng thay cho hằng có nhiều điểm thuận lợi như sau:

- Chương trình dễ đọc hơn, vì thay cho các con số ít có ý nghĩa, một tên gọi sẽ làm NSD dễ hình dung vai trò, nội dung của nó. Ví dụ, khi gặp tên gọi SOSV NSD sẽ hình dung được chẳng hạn, "đây là số sinh viên tối đa trong một lớp", trong khi số 50 có thể là số sinh viên mà cũng có thể là tuổi của một sinh viên nào đó.
- Chương trình dễ sửa chữa hơn, ví dụ bây giờ nếu muốn thay đổi chương trình sao cho bài toán quản lý được thực hiện với số sinh viên tối đa là 60, khi đó ta cần tìm và thay thế hàng trăm vị trí xuất hiện của 50 thành 60. Việc thay thế như vậy dễ gây ra lỗi vì có thể không tìm thấy hết các số 50 trong chương trình hoặc thay nhầm số 50 với ý nghĩa khác như tuổi của một sinh viên nào đó chẳng hạn. Nếu trong chương trình sử dụng hằng SOSV, bây giờ việc thay thế trở nên chính xác và dễ dàng hơn bằng thao tác khai báo lại giá trị hằng SOSV bằng 60. Lúc đó trong chương trình bất kỳ nơi nào gặp tên hằng SOSV đều được chương trình hiểu với giá trị 60.

Để khai báo hằng ta dùng các câu khai báo sau:

#define tên_hằng giá_trị_hằng ;

hoặc:

const tên_hằng = giá_trị_hằng ;

Ví dụ:

```
#define sosv 50 ;
#define MAX 100 ;
const sosv = 50 ;
```

Như trên đã chú ý một giá trị hằng chưa nói lên kiểu sử dụng của nó vì vậy ta cần khai báo rõ ràng hơn bằng cách thêm tên kiểu trước tên hằng trong khai báo const, các hằng khai báo như vậy được gọi là hằng có kiểu.

Ví dụ:

```
const int sosv = 50 ;
const float nhiet_do_soi = 100.0 ;
```

III. BIẾN - KHAI BÁO VÀ SỬ DỤNG BIẾN

1. Khai báo biến

Biến là các tên gọi để lưu giá trị khi làm việc trong chương trình. Các giá trị được lưu có thể là các giá trị dữ liệu ban đầu, các giá trị trung gian tạm thời trong quá trình tính toán hoặc các giá trị kết quả cuối cùng. Khác với hằng, giá trị của biến có thể thay đổi trong quá trình làm việc bằng các lệnh đọc vào từ bàn phím hoặc gán. Hình ảnh cụ thể của biến là một số ô nhớ trong bộ nhớ được sử dụng để lưu các giá trị của biến.

Mọi biến phải được khai báo trước khi sử dụng. Một khai báo như vậy sẽ báo cho chương trình biết về một biến mới gồm có: tên của biến, kiểu của biến (tức kiểu của giá trị dữ liệu mà biến sẽ lưu giữ). Thông thường với nhiều NNLT tất cả các biến phải được khai báo ngay từ đầu chương trình hay đầu của hàm, tuy nhiên để thuận tiện C++ cho phép khai báo biến ngay bên trong chương trình hoặc hàm, có nghĩa bất kỳ lúc nào NSD thấy cần thiết sử dụng biến mới, họ có quyền khai báo và sử dụng nó từ đó trở đi.

Cú pháp khai báo biến gồm tên kiểu, tên biến và có thể có hay không khởi tạo giá trị ban đầu cho biến. Để khởi tạo hoặc thay đổi giá trị của biến ta dùng lệnh gán (=).

a. Khai báo không khởi tạo

```
tên_kiểu tên_biến_1 ;
tên_kiểu tên_biến_2 ;
```

```
tên_kiểu  tên_biến_3 ;
```

Nhiều biến cùng kiểu có thể được khai báo trên cùng một dòng:

```
tên_kiểu  tên_biến_1, tên_biến_2, tên_biến_3 ;
```

Ví dụ:

```
void main()
{
    int i, j ;                  // khai báo 2 biến i, j có kiểu nguyên
    float x ;                   // khai báo biến thực x
    char c, d[100] ;            // biến kí tự c, xâu d chứa tối đa 100 kí tự
    unsigned int u ;            // biến nguyên không dấu u
    …
}
```

b. Khai báo có khởi tạo

Trong câu lệnh khai báo, các biến có thể được gán ngay giá trị ban đầu bởi phép toán gán (=) theo cú pháp:

tên_kiểu tên_biến_1 = gt_1, tên_biến_2 = gt_2, tên_biến_3 = gt_3 ;

trong đó các giá trị gt_1, gt_2, gt_3 có thể là các hằng, biến hoặc biểu thức.

Ví dụ:

```
const int n = 10 ;
void main()
{
    int i = 2, j , k = n + 5;       // khai báo i và khởi tạo bằng 2, k bằng 15
    float eps = 1.0e-6 ;            // khai báo biến thực epsilon khởi tạo bằng 10^-6
    char c = 'Z';                   // khai báo biến kí tự c và khởi tạo bằng 'A'
    char d[100] = "Tin học";        // khai báo xâu kí tự d chứa dòng chữ "Tin học"
    …
}
```

2. Phạm vi của biến

Như đã biết chương trình là một tập hợp các hàm, các câu lệnh cũng như các khai báo. Phạm vi tác dụng của một biến là nơi mà biến có tác dụng, tức hàm nào, câu lệnh

nào được phép sử dụng biến đó. Một biến xuất hiện trong chương trình có thể được sử dụng bởi hàm này nhưng không được bởi hàm khác hoặc bởi cả hai, điều này phụ thuộc chặt chẽ vào vị trí nơi biến được khai báo. Một nguyên tắc đầu tiên là biến sẽ có tác dụng kể từ vị trí nó được khai báo cho đến hết khối lệnh chứa nó. Chi tiết cụ thể hơn sẽ được trình bày trong chương 4 khi nói về hàm trong C++.

3. Gán giá trị cho biến (phép gán)

Trong các ví dụ trước chúng ta đã sử dụng phép gán dù nó chưa được trình bày, đơn giản một phép gán mang ý nghĩa tạo giá trị mới cho một biến. Khi biến được gán giá trị mới, giá trị cũ sẽ được tự động xoá đi bất kể trước đó nó chứa giá trị nào (hoặc chưa có giá trị, ví dụ chỉ mới vừa khai báo xong). Cú pháp của phép gán như sau:

tên_biến = biểu thức ;

Khi gặp phép gán chương trình sẽ tính toán giá trị của biểu thức sau đó gán giá trị này cho biến. Ví dụ:

```
int n, i = 3;                          // khởi tạo i bằng 3
n = 10;                                // gán cho n giá trị 10
cout << n <<", " << i << endl;         // in ra: 10, 3
i = n / 2;                             // gán lại giá trị của i bằng n/2 = 5
cout << n <<", " << i << endl;         // in ra: 10, 5
```

Trong ví dụ trên n được gán giá trị bằng 10; trong câu lệnh tiếp theo biểu thức n/2 được tính (bằng 5) và sau đó gán kết quả cho biến i, tức i nhận kết quả bằng 5 dù trước đó nó đã có giá trị là 2 (trong trường hợp này việc khởi tạo giá trị 2 cho biến i là không có ý nghĩa).

Một khai báo có khởi tạo cũng tương đương với một khai báo và sau đó thêm lệnh gán cho biến (ví dụ int i = 3 cũng tương đương với 2 câu lệnh int i; i = 3) tuy nhiên về mặt bản chất khởi tạo giá trị cho biến vẫn khác với phép toán gán như ta sẽ thấy trong các phần sau.

4. Một số điểm lưu ý về phép gán

Với ý nghĩa thông thường của phép toán (nghĩa là tính toán và cho lại một giá trị) thì phép toán gán còn một nhiệm vụ nữa là trả lại một giá trị. Giá trị trả lại của phép toán gán chính là giá trị của biểu thức sau dấu bằng. Lợi dụng điều này C++ cho phép chúng ta gán "kép" cho nhiều biến nhận cùng một giá trị bởi cú pháp:

biến_1 = biến_2 = … = biến_n = gt ;

với cách gán này tất cả các biến sẽ nhận cùng giá trị gt. Ví dụ:

```
int i, j, k ;
i = j = k = 1;
```

Biểu thức gán trên có thể được viết lại như (i = (j = (k = 1))), có nghĩa đầu tiên để thực hiện phép toán gán giá trị cho biến i chương trình phải tính biểu thức (j = (k = 1)), tức phải tính k = 1, đây là phép toán gán, gán giá trị 1 cho k và trả lại giá trị 1, giá trị trả lại này sẽ được gán cho j và trả lại giá trị 1 để tiếp tục gán cho i.

Ngoài việc gán kép như trên, phép toán gán còn được phép xuất hiện trong bất kỳ biểu thức nào, điều này cho phép trong một biểu thức có phép toán gán, nó không chỉ tính toán mà còn gán giá trị cho các biến, ví dụ n = 3 + (i = 2) sẽ cho ta i = 2 và n = 5. Việc sử dụng nhiều chức năng của một câu lệnh làm cho chương trình gọn gàng hơn (trong một