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 số trường hợp) nhưng cũng trở nên khó đọc, chẳng hạn câu lệnh trên có thể viết tách thành 2 câu lệnh i = 2; n = 3 + i; sẽ dễ đọc hơn ít nhất đối với các bạn mới bắt đầu tìm hiểu về lập trình.

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

hơn cạnh còn lại). Tính chu vi, diện tích, độ dài 3 đường cao, 3 đường trung tuyến, 3 đường phân giác, bán kính đường tròn nội tiếp, ngoại tiếp lần lượt theo các công thức sau:

C = 2p = a + b + c ; S = $\sqrt{p(p-a)(p-b)(p-c)}$;

$h_a = \frac{2S}{a}$; ma = $\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$; ga = $\frac{2}{b+c}\sqrt{bcp(p-a)}$;

$r = \frac{S}{p}$; $R = \frac{abc}{4S}$;

14. Tính diện tích và thể tích của hình cầu bán kính R theo công thức:

S = $4\pi R^2$; V = RS/3

15. Nhập vào 4 chữ số. In ra tổng của 4 chữ số này và chữ số hàng chục, hàng đơn vị của tổng (ví dụ 4 chữ số 3, 1, 8, 5 có tổng là 17 và chữ số hàng chục là 1 và hàng đơn vị là 7, cần in ra 17, 1, 7).

16. Nhập vào một số nguyên (có 4 chữ số). In ra tổng của 4 chữ số này và chữ số đầu, chữ số cuối (ví dụ số 3185 có tổng các chữ số là 17, đầu và cuối là 3 và 5, kết quả in ra là: 17, 3, 5).

17. Hãy nhập 2 số a và b. Viết chương trình đổi giá trị của a và b theo 2 cách:

- dùng biến phụ t: t = a; a = b; b = t;
- không dùng biến phụ: a = a + b; b = a - b; a = a - b;

In kết quả ra màn hình để kiểm tra.

18. Viết chương trình đoán số của người chơi đang nghĩ, bằng cách yêu cầu người chơi nghĩ một số, sau đó thực hiện một loạt các tính toán trên số đã nghĩ rồi cho biết kết quả. Máy sẽ in ra số mà người chơi đã nghĩ. (ví dụ yêu cầu người chơi lấy số đã nghĩ nhân đôi, trừ 4, bình phương, chia 2 và trừ 7 rồi cho biết kết quả, máy sẽ in ra số người chơi đã nghĩ).

19. Một sinh viên gồm có các thông tin: họ tên, tuổi, điểm toán (hệ số 2), điểm tin (hệ số 1). Hãy nhập các thông tin trên cho 2 sinh viên. In ra bảng điểm gồm các chi tiết nêu trên và điểm trung bình của mỗi sinh viên.

20. Một nhân viên gồm có các thông tin: họ tên, hệ số lương, phần trăm phụ cấp (theo lương) và phần trăm phải đóng BHXH. Hãy nhập các thông tin trên cho 2 nhân viên. In ra bảng lương gồm các chi tiết nêu trên và tổng số tiền cuối cùng mỗi nhân viên được nhận.