TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

----------o0o----------

BÀI GIẢNG NHẬP MÔN LẬP TRÌNH

Nhóm biên soạn: Bộ môn Công nghệ lập trình &Ứng dụng

Hệ đào tạo: Đại học chính qui

Năm học 2015 – 2016

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

----------o0o----------

BÀI GIẢNG MÔN HỌC NHẬP MÔN LẬP TRÌNH

HỆ ĐẠI HỌC CHÍNH QUY

KHOA PHÊ DUYỆT BM PHÊ DUYỆT GV PHỤC TRÁCH

Năm học 2015 – 2016

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU TỔNG QUAN VỀ LẬP TRÌNH

1.1. Khái niệm về chương trình máy tính

Phần này chúng ta sẽ tìm hiểu một số khái niệm căn bản về thuật toán, chương trình, ngôn ngữ lập trình. Thuật ngữ "thuật giải" và "thuật toán" dĩ nhiên có sự khác nhau song trong nhiều trường hợp chúng có cùng nghĩa.

1.1.1. Chương trình (Program)

Là một tập hợp các mô tả, các phát biểu, nằm trong một hệ thống qui ước về ý nghĩa và thứ tự thực hiện, nhằm điều khiển máy tính làm việc. Theo Niklaus Wirth thì:

Chương trình = Thuật toán + Cấu trúc dữ liệu

Các thuật toán và chương trình đều có cấu trúc dựa trên ***3 cấu trúc điều khiển cơ bản***:

Tuần tự (Sequential): Các bước thực hiện tuần tự một cách chính xác từ trên xuống, mỗi bước chỉ thực hiện đúng một lần.

Chọn lọc (Selection): Chọn 1 trong 2 hay nhiều thao tác để thực hiện.

Lặp lại (Repetition): Một hay nhiều bước được thực hiện lặp lại một số lần.

Muốn trở thành lập trình viên chuyên nghiệp bạn hãy làm đúng trình tự để có thói quen tốt và thuận lợi sau này trên nhiều mặt của một người làm máy tính. Bạn hãy làm theo các bước sau:

Tìm, xây dựng thuật giải (trên giấy) → viết chương trình trên máy

→ dịch chương trình → chạy và thử chương trình

1.1.2. Giải thuật (Algorithm)

Là một dãy các thao tác xác định trên một đối tượng, sao cho sau khi thực hiện một số hữu hạn các bước thì đạt được mục tiêu. Theo R.A.Kowalski thì bản chất của thuật giải:

Thuật giải = Logic + Điều khiển

Logic: Đây là phần khá quan trọng, nó trả lời câu hỏi "Thuật giải làm gì, giải quyết vấn đề gì?", những yếu tố trong bài toán có quan hệ với nhau như thế nào v.v… Ở đây bao gồm những kiến thức chuyên môn mà bạn phải biết để có thể tiến hành giải bài toán.

Ví dụ 1.1: Để giải một bài toán tính diện tích hình cầu, mà bạn không còn nhớ công thức tính hình cầu thì bạn không thể viết chương trình cho máy để giải bài toán này được.

Điều khiển: Thành phần này trả lời câu hỏi: giải thuật phải làm như thế nào?. Chính là cách thức tiến hành áp dụng thành phần logic để giải quyết vấn đề.

1.1.3. Ngôn ngữ lập trình (Programming language)

Ngôn ngữ lập trình là hệ thống các ký hiệu tuân theo các qui ước về ngữ pháp và ngữ nghĩa, dùng để xây dựng thành các chương trình cho máy tính.

Một chương trình được viết bằng một ngôn ngữ lập trình cụ thể (ví dụ C, …) gọi là chương trình nguồn, chương trình dịch làm nhiệm vụ dịch chương trình nguồn thành chương trình thực thi được trên máy tính.

1.2. Biểu diễn thuật toán

Khi chứng minh hoặc giải một bài toán trong toán học, ta thường dùng những ngôn từ toán học như: "ta có", "điều phải chứng minh", "giả thuyết", ... và sử dụng những phép suy luận toán học như phép suy ra, tương đương, ...Thuật toán là một phương pháp thể hiện lời giải bài toán nên cũng phải tuân theo một số quy tắc nhất định. Để có thể truyền đạt thuật toán cho người khác hay chuyển thuật toán thành chương trình máy tính, ta phải có phương pháp biểu diễn thuật toán.

a) Các đặc trưng của thuật toán:

Tính xác định: Các thao tác, các đối tượng, phương tiện trong thuật toán phải có ý nghĩa rõ ràng, không được gây nhầm lẫn. Nói cách khác, hai cơ chế hoạt động khác nhau cùng thực hiện một thuật toán, sử dụng các đối tượng, phương tiện nhập phải cho cùng một kết quả.

Tính dừng: Đòi hỏi thuật toán phải dừng và cho kết quả sau một số hữu hạn các bước.

Tính đúng của thuật toán: Thuật toán đúng là thuật toán cho kết quả thỏa mãn đặc tả thuật toán với mọi trường hợp của các đối tượng, phương tiện nhập.

Tính phổ dụng: Thuật toán để giải một lớp bài toán gồm nhiều bài cụ thể, lớp đó được xác định bởi đặc tả. Dĩ nhiên là có lớp bài toán chỉ gồm 1 bài. Thuật toán khi đó sẽ không cần sử dụng đối tượng, phương tiện nhập nào cả.

b) Phương pháp biểu diễn:

Thuật toán có thể diễn đạt dưới nhiều hình thức, chẳng hạn dưới dạng lưu đồ, dạng ngôn ngữ tự nhiên, dạng mã giả hoặc một ngôn ngữ lập trình nào khác.

Dạng ngôn ngữ tự nhiên: Thuật toán có thể trình bày dưới dạng ngôn ngữ tự nhiên theo trình tự các bước thực hiện trong thuật toán.

Dạng ngôn ngữ lập trình: Dùng cấu trúc lệnh, dữ liệu của một ngôn ngữ lập trình nào đó để mô tả.

Dạng mã giả: Thuật toán trình bày trong dạng văn bản bằng ngôn ngữ tự nhiên tuy dễ hiểu nhưng khó cài đặt. Dùng một ngôn ngữ lập trình nào đó để diễn tả thì phức tạp, khó hiểu. Thông thường thuật toán cũng được trao đổi dưới dạng văn bản – tuy không ràng buộc nhiều vào cú pháp xác định như các ngôn ngữ lập trình, nhưng cũng tuân theo một số quy ước ban đầu – ta gọi là dạng mã giả. Tùy theo việc định hướng cài đặt thuật toán theo ngôn ngữ lập trình nào ta diễn đạt thuật toán gần với ngôn ngữ ấy.

Dạng lưu đồ: Trong các phương pháp biểu diễn, chúng ta sẽ chủ yếu nghiên cứu phương pháp biểu diễn theo dạng này. Dạng lưu đồ dùng các hình vẽ (có quy ước) để diễn đạt thuật toán. Lưu đồ cho hình ảnh trực quan và tổng thể của thuật toán, cho nên thường được sử dụng nhiều nhất.

1.2.1. Sử dụng ngôn ngữ tự nhiên

Trong cách biểu diễn thuật toán theo ngôn ngữ tự nhiên, người ta sử dụng ngôn ngữ thường ngày để liệt kê các bước của thuật toán. Phương pháp biểu diễn này *không*

yêu cầu người viết thuật toán cũng như người đọc thuật toán phải nắm các quy tắc. Tuy vậy, cách biểu diễn này thường dài dòng, không thể hiện rõ cấu trúc của thuật toán, đôi lúc gây hiểu lầm hoặc khó hiểu cho người đọc. Gần như không có một quy tắc cố định nào trong việc thể hiện thuật toán bằng ngôn ngữ tự nhiên. Tuy vậy, để dễ đọc, ta nên viết các bước con lùi vào bên phải và đánh số bước theo quy tắc phân cấp như 1, 1.1, 1.1.1, ...

Ví dụ 1.2: Để tính tổng các số nguyên dương lẻ trong khoảng từ 1 đến n ta có thuật toán sau:

B1. Hỏi giá trị của n.

B2. S = 0

B3. i = 1

B4. Nếu i = n+1 thì sang bước B8, ngược lại sang bước B5

B5. Cộng thêm i vào S

B6. Cộng thêm 2 vào i

B7. Quay lại bước B4.

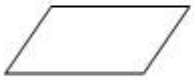
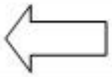
B8. Tổng cần tìm chính là S.

Ta chú ý đến bước B4. Ở đây ta muốn kết thúc thuật toán khi giá trị của i vượt quá n. Thay vì viết là "nếu i lớn hơn n" thì ta thay bằng điều kiện "nếu i bằng n+1" vì theo toán học "i = n+1" thì suy ra "i lớn hơn n". Nhưng điều kiện "i = n+1" không phải lúc nào cũng đạt được. Vì ban đầu i = 1 là số lẻ, sau mỗi bước, i được tăng thêm 2 nên i luôn là số lẻ. Nếu n là số chẵn thì n+1 là một số lẻ nên sau một số bước nhất định, i sẽ bằng n+1. Tuy nhiên, nếu n là một số lẻ thì n+1 là một số chẵn, do i là số lẻ nên dù có qua bao nhiêu bước đi chăng nữa, i vẫn khác n+1. Trong trường hợp đó, thuật toán trên sẽ bị quẩn.

1.2.2. Sử dụng lưu đồ - Sơ đồ khối

Để dễ hơn về quy trình xử lý, các nhà lập trình đưa ra dạng lưu đồ để minh họa từng bước quá trình xử lý một vấn đề (bài toán).

Các ký hiệu sử dụng trong phương pháp biểu diễn thuật toán bằng lưu đồ:

STT	Ký hiệu	Giải thích
1		Bắt đầu và kết thúc chương trình
2		Điểm nối, đường đi (luồng xử lý)
3		Điều khiển lựa chọn
4		Thao tác nhập, xuất.
5		Thao tác xử lý hoặc tính toán.
6		Trả về giá trị (return)
7		Điểm nối liên kết tiếp theo (sử dụng khi lưu đồ vượt quá trang)

Chú ý khi vẽ lưu đồ:

- Trước tiên hãy tập trung vẽ một số đường đi chính của lưu đồ.

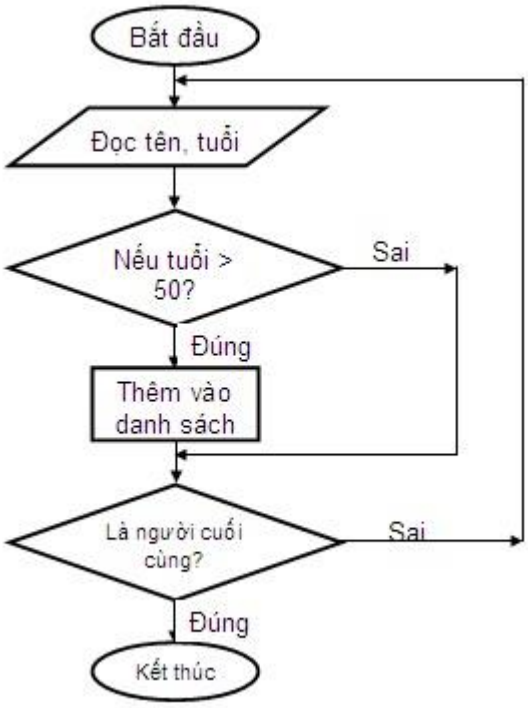
- Thêm vào tất cả các nhánh và vòng lặp.

- Một lưu đồ chỉ có một điểm Bắt đầu và một điểm kết thúc.

- Mỗi bước trong chương trình không cần thể hiện trong lưu đồ.

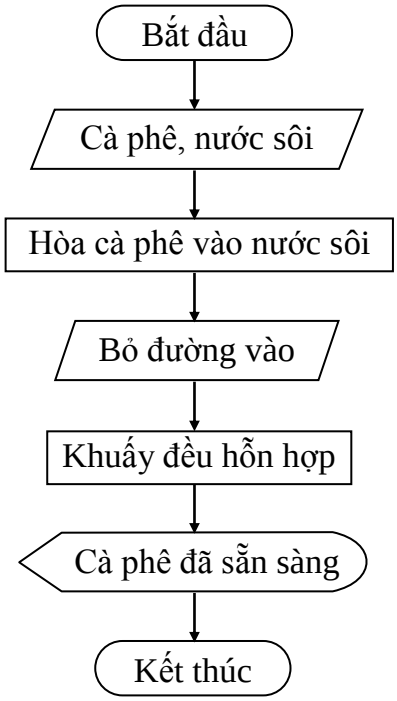
- Lưu đồ cần phải đáp ứng được yêu cầu: những người lập trình khác có thể hiểu lưu đồ một cách dễ dàng.

Ví dụ 1.3: Đọc các thông tin như tên, tuổi và lưu lại những người có tuổi trên 50

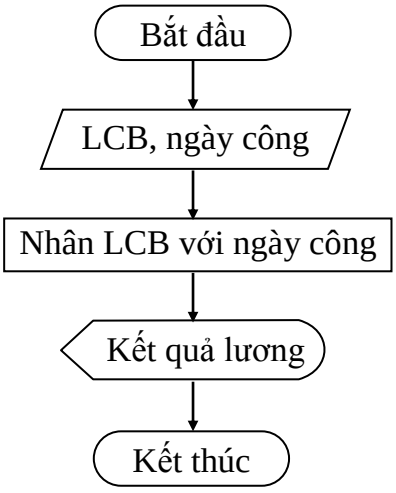


Ví dụ 1.4:

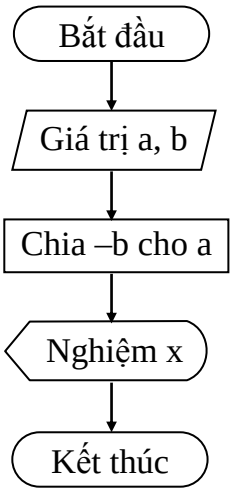
Chuẩn bị cà phê



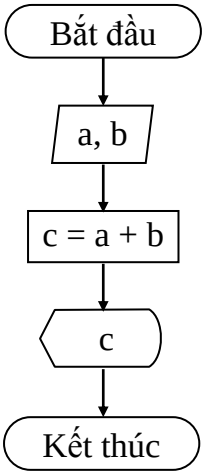
Mô tả Ví dụ



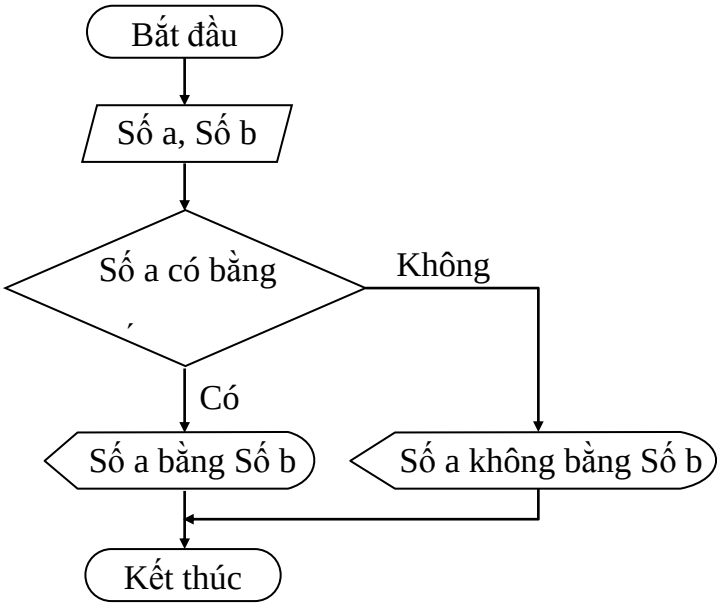
Mô tả Ví dụ



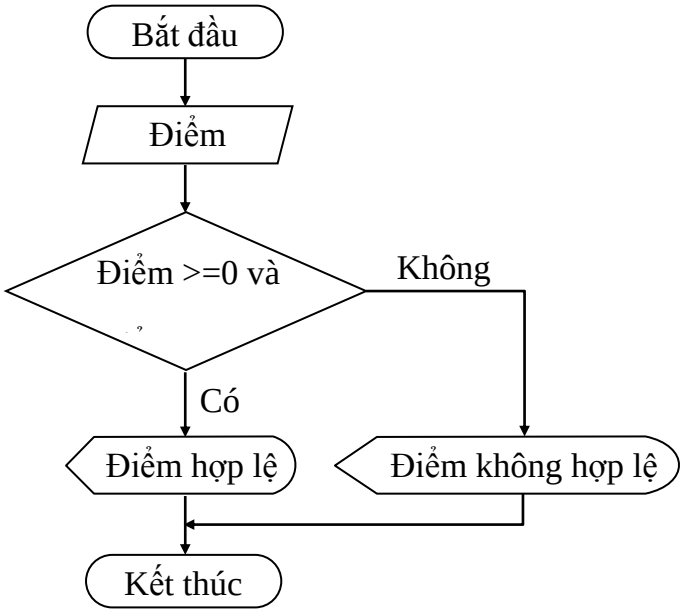
Cộng 2 số



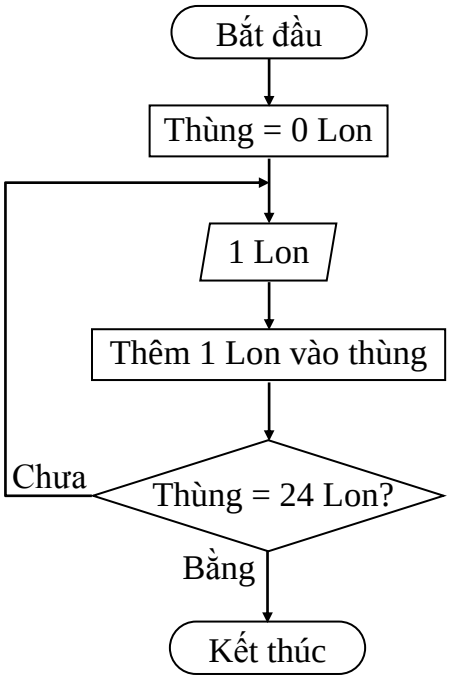
So sánh 2 số



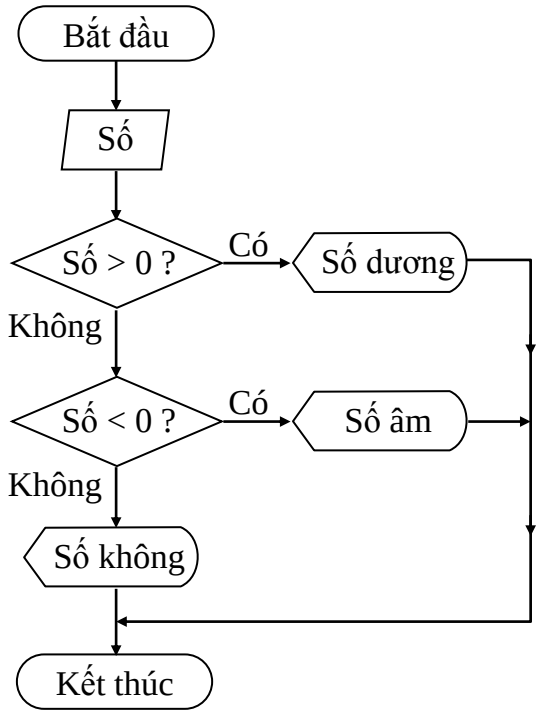
Kiểm tra tính hợp lệ của điểm



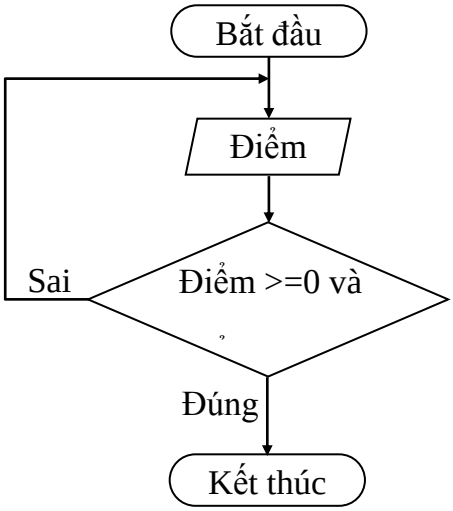
Xếp lon vào thùng



Kiểm tra loại số



Kiểm tra tính hợp lệ của điểm



1.2.3. Sử dụng mã giả

Tuy sơ đồ khối thể hiện rõ quá trình xử lý và sự phân cấp các trường hợp của thuật toán nhưng lại cồng kềnh. Để mô tả một thuật toán nhỏ ta phải dùng một không gian rất lớn. Hơn nữa, lưu đồ chỉ phân biệt hai thao tác là rẽ nhánh (chọn lựa có điều kiện) và xử lý mà trong thực tế, các thuật toán còn có thêm các thao tác lặp.

Khi thể hiện thuật toán bằng mã giả, ta sẽ *vay mượn* các cú pháp của một ngôn ngữ lập trình nào đó để thể hiện thuật toán. Tất nhiên, mọi ngôn ngữ lập trình đều có những thao tác cơ bản là xử lý, rẽ nhánh và lặp. Dùng mã giả vừa tận dụng được các khái niệm trong ngôn ngữ lập trình, vừa giúp người cài đặt dễ dàng nắm bắt nội dung thuật toán. Tất nhiên là trong mã giả ta vẫn dùng một phần ngôn ngữ tự nhiên. Một khi đã vay mượn cú pháp và khái niệm của ngôn ngữ lập trình thì chắc chắn mã giả sẽ bị phụ thuộc vào ngôn ngữ lập trình đó. Chính vì lý do này, chúng ta chưa vội tìm hiểu về mã giả trong bài này (vì chúng ta chưa biết gì về ngôn ngữ lập trình!). Sau khi tìm hiểu xong bài về thủ tục - hàm bạn sẽ hiểu mã giả là gì !

Một đoạn mã giả của thuật toán giải phương trình bậc hai:

```
if delta > 0 then
begin
```

```
     x1 = (-b-sqrt(delta))/(2*a)
     x2 = (-b+sqrt(delta))/(2*a)
     xuất kết quả: phương trình có hai nghiệm là x1 và x2
end
else
     if delta = 0 then
          xuất kết quả: phương trình có nghiệm kép là -
b/(2*a)
     else {trường hợp delta < 0 }
          xuất kết quả: phương trình vô nghiệm
```

1.3. Các bước xây dựng chương trình

Bước 1: Phân tích vấn đề và xác định các đặc điểm. (xác định I-P-O)

Bước 2: Lập ra giải pháp. (đưa ra thuật giải)

Bước 3: Cài đặt. (viết chương trình)

Bước 4: Chạy thử chương trình. (dịch chương trình)

Bước 5: Kiểm chứng và hoàn thiện chương trình. (thử nghiệm bằng nhiều số liệu và đánh giá)

1.4. Hệ đếm và biểu diễn số trong hệ đếm

1.4.1. Hệ đếm

Hệ đếm là tập hợp các ký hiệu và qui tắc sử dụng tập ký hiệu đó để biểu diễn và xác định các giá trị các số. Mỗi hệ đếm có một số ký số hữu hạn. Tổng số ký số của mỗi hệ đếm được gọi là cơ số (base hay radix), ký hiệu là b.

Trong ngành toán - tin học hiện nay phổ biến 4 hệ đếm như sau:

Hệ đếm	Cơ số	Ký số và trị tuyệt đối
Hệ nhị phân	2	0, 1
Hệ bát phân	8	0, 1, 2, 3, 4, 5, 6, 7
Hệ thập phân	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hệ thập lục phân	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Hệ đếm phổ biến hiện nay là hệ đếm thập phân.

1.4.2. Biểu diễn số trong các hệ đếm

1.4.2.1. Hệ đếm thập phân (decimal system)

Hệ đếm thập phân hay hệ đếm cơ số 10 là một trong những phát minh của người Ả rập cổ, bao gồm 10 ký số theo ký hiệu sau:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Qui tắc tính giá trị của hệ đếm này là mỗi đơn vị ở một hàng bất kỳ có giá trị bằng 10 đơn vị của hàng kế cận bên phải. Ở đây b = 10. Bất kỳ số nguyên dương trong hệ thập phân có thể thể hiện như là một tổng các chuỗi các ký số thập phân nhân cho 10 lũy thừa, trong đó số mũ lũy thừa được tăng thêm 1 đơn vị kể từ số mũ lũy thừa phía bên phải nó. Số mũ lũy thừa của hàng đơn vị trong hệ thập phân là 0.

Ví dụ 1.5: Số 2105 có thể được thể hiện như sau:

$2165 = 2 \times 10^3 + 1 \times 10^2 + 6 \times 10^1 + 5 \times 10^0$

$= 2 \times 1000 + 1 \times 100 + 6 \times 10 + 5 \times 1$

Thể hiện như trên gọi là ký hiệu mở rộng của số nguyên vì:

2165 = 2000+100 +60+5

Như vậy, trong số 2165: ký số 5 trong số nguyên đại diện cho giá trị 5 đơn vị (1s), ký số 6 đại diện cho giá trị 6 chục (10s), ký số 1 đại diện cho giá trị 1 trăm (100s) và ký số 2 đại diện cho giá trị 2 nghìn (1000s). Nghĩa là, số lũy thừa của 10 tăng dần 1 đơn vị từ trái sang phải tương ứng với vị trí ký hiệu số,

$10^0 = 1$ $10^1 = 10$ $10^2 = 100$ $10^3 = 1000$ $10^4 = 10000$ ...

Mỗi ký số ở thứ tự khác nhau trong số sẽ có giá trị khác nhau, ta gọi là giá trị vị trí (place value).

Phần phân số trong hệ thập phân sau dấu chấm phân cách (theo qui ước của Mỹ) thể hiện trong ký hiệu mở rộng bởi 10 lũy thừa âm tính từ phải sang trái kể từ dấu chấm phân cách:

Ví dụ 1.6:

$2165.37 = 2 \times 10^3 + 1 \times 10^2 + 6 \times 10^1 + 5 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2}$

$= 2 \times 1000 + 1 \times 100 + 6 \times 10 + 5 \times 1 + 3 \times \frac{1}{10} + 7 \times \frac{1}{100}$

$= 2000 + 100 + 60 + 5 + \frac{3}{10} + \frac{7}{100}$

Tổng quát, hệ đếm cơ số b (b≥2, b là số nguyên dương) mang tính chất sau:

- ✓ Có b ký số để thể hiện giá trị số. Ký số nhỏ nhất là 0 và lớn nhất là b-1.
- ✓ · Giá trị vị trí thứ n trong một số của hệ đếm bằng cơ số b lũy thừa n : b

Số N(b) trong hệ đếm cơ số (b) thể hiện :

$N_{(b)} = a_n a_{n-1} a_{n-2} \ldots a_1 a_0 a_{-1} a_{-2} \ldots a_{-m}$

trong đó, số N(b) có n+1 ký số ở phần nguyên và m ký số ở phần thập phân, sẽ có giá trị là :

$N_{(b)} = a_n \times b^n + a_{n-1} \times b^{n-1} + a_{n-2} \times b^{n-2} \ldots a_1 \times b^1 + a_0 \times b^0 + a_{-1} \times b^{-1} + a_{-2} \times b^{-2} \ldots a_{-m} \times b^{-m}$

Hay: $N_{(b)} = \sum_{i=-m}^{n} a_i b^i$

1.4.3. Hệ đếm nhị phân (binary number system)

Với b=2, chúng ta có hệ đếm nhị phân. Đây là hệ đếm đơn giản nhất với 2 chữ số là 0 và 1. Mỗi chữ số nhị phân gọi là BIT (viết tắt từ chữ BInary digiT). Hệ nhị phân tương ứng với 2 trạng thái của các linh kiện điện tử trong máy tính chỉ có: đóng (có điện hay có dòng điện đi qua) ký hiệu là 1 và tắt (không có điện hay không có dòng điện đi qua) ký hiệu là 0. Vì hệ nhị phân chỉ có 2 trị số là 0 và 1, nên khi muốn diễn tả một số lớn hơn, hoặc các ký tự phức tạp hơn thì cần kết hợp nhiều bit với nhau.

Ta có thể chuyển đổi hệ nhị phân theo hệ thập phân quen thuộc.

Ví dụ 1.7: Số $1110101_{(2)}$ sẽ tương đương với giá trị thập phân là: 117

Số nhị phân	1	1	1	0	1	0	1

Vị trí	6	5	4	3	2	1	0
Trị vị trí	26	25	24	23	22	21	20
Hệ 10 là	64	32	16	8	4	2	1

Như vậy: $1110101_{(2)} = 1x64 + 1x32 + 1x\ 16 + 0x8 + 1x4 + 0x2 + 1x1 = 117_{(10)}$

1.4.4. Hệ đếm thập lục phân (hexa-decimal number system)

Hệ đếm thập lục phân là hệ cơ số b = 16 =2^4, tương đương với tập 4 chữ số nhị phân (4 bit). Khi thể hiện ở dạng hexa-decimal, ta có 16 ký tự gồm 10 chữ số từ 0 đến 9, và 6 chữ in A, B, C, D, E, F để biểu diễn các giá trị số tương ứng là 10, 11, 12, 13, 14, 15. Với hệ thập lục phân, trị vị trí là lũy thừa của 16.

Ví dụ 1.8:

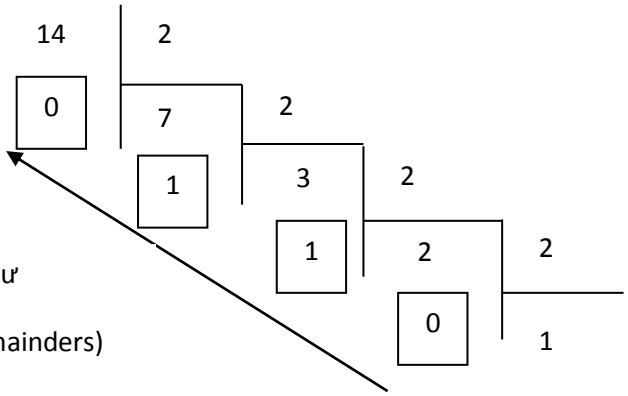
$75_{(16)} = 7x16^1 + 5x16^0 = 117_{(10)}$

$A2B_{(16)}=10x16^2 + 2x16^1 + 11x16^0 = 2603_{(10)}$

1.4.5. Đổi một số nguyên từ hệ thập phân sang hệ b

Tổng quát: Lấy số nguyên thập phân N(10) lần lượt chia cho b cho đến khi thương số bằng 0. Kết quả số chuyển đổi N(b) là các dư số trong phép chia viết ra theo thứ tự ngược lại.

Ví dụ 1.9: Số 14 trong hệ thập phân sẽ được biểu diễn như thế nào trong hệ nhị phân (b =2). Dùng phép chia 2 liên tiếp ta có các số dư như sau:



Ta được: $14_{(10)} = 0110$

CHƯƠNG 2. CÁC THÀNH PHẦN TRONG NGÔN NGỮ C

2.1. Các khái niệm cơ bản

2.1.1. Từ khóa

Từ khoá là những từ được sử dụng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh. Bảng dưới đây liệt kê các từ khoá của TURBO C:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Ý nghĩa và cách sử dụng của mỗi từ khoá sẽ được đề cập sau này, ở đây ta cần *Chú ý:*

- Không được dùng các từ khoá để đặt tên cho các hằng, biến, mảng, hàm ...

- Từ khoá phải được viết bằng chữ thường, ví dụ: viết từ khoá khai báo kiểu nguyên là int chứ không phải là INT.

2.1.2. Tên

Tên là một khái niệm rất quan trọng, nó dùng để xác định các đại lượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên con trỏ, tên tệp, tên cấu trúc, tên nhãn, ...

Tên được đặt theo qui tắc sau:

Tên là một dãy các ký tự bao gồm chữ cái, số và gạch nối. Ký tự đầu tiên của tên phải là chữ hoặc gạch nối. Tên không được trùng với khoá. Độ dài cực đại của tên theo

mặc định là 32 và có thể được đặt lại là một trong các giá trị từ 1 tới 32 nhờ chức năng: Option-Compiler-Source-Identifier length khi dùng TURBO C.

Ví dụ 2.1:

Các tên đúng: a_1 delta x1 _step GAMA

Các tên sai:

3MN	Ký tự đầu tiên là số
m#2	Sử dụng ký tự #
f(x)	Sử dụng các dấu ()
do	Trùng với từ khoá
te ta	Sử dụng dấu trắng
Y-3	Sử dụng dấu -

Chú ý: Trong C, tên bằng chữ thường và chữ hoa là khác nhau ví dụ tên AB khác với ab. Trong C, ta thường dùng chữ hoa để đặt tên cho các hằng và dùng chữ thường để đặt tên cho hầu hết cho các đại lượng khác như biến, biến mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc.

2.1.3. Tập ký tự dùng trong ngôn ngữ C

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo nên các từ. Các từ lại được liên kết với nhau theo một qui tắc nào đó để tạo nên các câu lệnh. Một chương trình bao gồm nhiều câu lệnh và thể hiện một thuật toán để giải một bài toán nào đó. Ngôn ngữ C được xây dựng trên bộ ký tự sau:

- 26 chữ cái hoa: A B C .. Z

- 26 chữ cái thường: a b c .. z

- 10 chữ số: 0 1 2 .. 9

- Các ký hiệu toán học: + - * / = ()

- Ký tự gạch nối: _

- Các ký tự khác: . ,: ; [ ] {} ! \ & % # $ ...

Dấu cách (space) dùng để tách các từ. Ví dụ chữ VIET NAM có 8 ký tự, còn VIETNAM chỉ có 7 ký tự.

Chú ý:

Khi viết chương trình, ta không được sử dụng bất kỳ ký tự nào khác ngoài các ký tự trên.

Ví dụ như khi lập chương trình giải phương trình bậc hai $ax^2 + bx + c = 0$, ta cần tính biệt thức Delta $\Delta = b^2 - 4ac$, trong ngôn ngữ C không cho phép dùng ký tự Δ, vì vậy ta phải dùng ký hiệu khác để thay thế.

2.1.4. Các kiểu dữ liệu cơ sở

Trong C sử dụng các các kiểu dữ liệu sau:

a) Kiểu ký tự (char)

Một giá trị kiểu char chiếm 1 byte (8 bit) và biểu diễn được một ký tự thông qua bảng mã ASCII. Ví dụ:

0	NUL	16	DLE	32	SPC	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Có hai kiểu dữ liệu char: kiểu signed char và unsigned char.

Kiểu	Phạm vi biểu diễn	Số ký tự	Kích thước
Char (Signed char)	-128 đến 127	256	1 byte
Unsigned char	0 đến 255	256	1 byte

Ví dụ sau minh hoạ sự khác nhau giữa hai kiểu dữ liệu trên: Xét đoạn chương trình sau:

```
char ch1;
```

unsigned char ch2;

.....

ch1 = 200; ch2 = 200;

Khi đó thực chất:

ch1 = -56;

ch2 = 200;

Nhưng cả ch1 và ch2 đều biểu diễn cùng một ký tự có mã 200.

Phân loại ký tự:

Có thể chia 256 ký tự làm ba nhóm:

Nhóm 1: Nhóm các ký tự điều khiển có mã từ 0 đến 31. Chẳng hạn ký tự mã 13 dùng để chuyển con trỏ về đầu dòng, ký tự 10 chuyển con trỏ xuống dòng dưới (trên cùng một cột). Các ký tự nhóm này nói chung không hiển thị ra màn hình.

Nhóm 2: Nhóm các ký tự văn bản có mã từ 32 đến 126. Các ký tự này có thể được đưa ra màn hình hoặc máy in.

Nhóm 3: Nhóm các ký tự đồ hoạ có mã số từ 127 đến 255. Các ký tự này có thể đưa ra màn hình nhưng không in ra được (bằng các lệnh DOS).

b) Kiểu nguyên

Trong C cho phép sử dụng số nguyên kiểu int, số nguyên dài kiểu long và số nguyên không dấu kiểu unsigned. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây:

Kiểu	Phạm vi biểu diễn	Kích thước
int	-32768 đến 32767	2 byte
unsigned int	0 đến 65535	2 byte
long	-2147483648 đến 2147483647	4 byte
unsigned long	0 đến 4294967295	4 byte

Chú ý:

Kiểu ký tự cũng có thể xem là một dạng của kiểu nguyên.

c) Kiểu dấu phảy động

Trong C cho phép sử dụng ba loại dữ liệu dấu phảy động, đó là float, double và long double. Kích cỡ và phạm vi biểu diễn của chúng được chỉ ra trong bảng dưới đây:

Kiểu	Phạm vi biểu diễn	Số chữ số có nghĩa	Kích thước
Float	3.4E-38 đến 3.4E+38	7 đến 8	4 byte
Double	1.7E-308 đến 1.7E+308	15 đến 16	8 byte
long double	3.4E-4932 đến 1.1E4932	17 đến 18	10 byte

Giải thích: Máy tính có thể lưu trữ được các số kiểu float có giá trị tuyệt đối từ 3.4E-38 đến 3.4E+38. Các số có giá trị tuyệt đối nhỏ hơn3.4E-38 được xem bằng 0. Phạm vi biểu diễn của số double được hiểu theo nghĩa tương tự.

d) Định nghĩa kiểu bằng typedef

Công dụng: Từ khoá typedef dùng để đặt tên cho một kiểu dữ liệu. Tên kiểu sẽ được dùng để khai báo dữ liệu sau này. Nên chọn tên kiểu ngắn và gọn để dễ nhớ. Chỉ cần thêm từ khoá typedef vào trước một khai báo ta sẽ nhận được một tên kiểu dữ liệu và có thể dùng tên này để khai báo các biến, mảng, cấu trúc, vv...

Cách viết: Viết từ khoá typedef, sau đó kiểu dữ liệu (một trong các kiểu trên), rồi đến tên của kiểu.

Ví dụ 2.2:

typedef int nguyen;

sẽ đặt tên một kiểu int là nguyen. Sau này ta có thể dùng kiểu nguyen để khai báo các biến, các mảng int như ví dụ sau:

nguyen x, y, a[10], b[20][30];

Tương tự cho các câu lệnh:

typedef float mt50[50];

Đặt tên một kiểu mảng thực một chiều có 50 phần tử tên là mt50.

typedef int m_20_30[20][30];

Đặt tên một kiểu mảng thực hai chiều có 20x30 phần tử tên là m_20_30.

Sau này ta sẽ dùng các kiểu trên khai báo:

mt50 a, b;

m_20_30 x, y;

e) Hằng

Hằng là các đại lượng mà giá trị của nó không thay đổi trong quá trình tính toán.

Tên hằng:

Để đặt tên một hằng, ta dùng dòng lệnh sau:

#define tên hằng giá trị

Ví dụ 2.3:

```
#define MAX 1000
```

Lúc này, tất cả các tên MAX trong chương trình xuất hiện sau này đều được thay bằng 1000. Vì vậy, ta thường gọi MAX là tên hằng, nó biểu diễn số 1000.

Một ví dụ khác:

```
#define pi 3.141593
```

Đặt tên cho một hằng float là pi có giá trị là 3.141593.

Các loại hằng:

- **Hằng int:** Hằng int là số nguyên có giá trị trong khoảng từ -32768 đến 32767.

Ví dụ 2.4:

```
#define number1 -50 //Định nghiã hằng int number1 có giá trị là -50
#define sodem 2732 //Định nghiã hằng int sodem có giá trị là 2732
```

Chú ý: Cần phân biệt hai hằng 5056 và 5056.0: ở đây 5056 là số nguyên còn 5056.0 là hằng thực.

- **Hằng long:** Hằng long là số nguyên có giá trị trong khoảng từ -2147483648 đến 2147483647.

Hằng long được viết theo cách:

1234L hoặc 1234l

(thêm L hoặc l vào đuôi)

Một số nguyên vượt ra ngoài miền xác định của int cũng được xem là long.

Ví dụ 2.5:

```
#define sl 8865056L          //Định nghiã hằng long sl có giá trị là 8865056
#define sl 8865056           //Định nghiã hằng long sl có giá trị là 8865056
```

- **Hằng ký tự:** Hằng ký tự là một ký tự riêng biệt được viết trong hai dấu nháy đơn, ví dụ 'a'.

Giá trị của 'a' chính là mã ASCII của chữ a. Như vậy giá trị của 'a' là 97. Hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác.

Ví dụ 2.6:

'9'-'0' = 57-48 = 9

```
#define kt 'a'          Định nghiã hằng ký tự kt có giá trị là 97
```

Đối với một vài hằng ký tự đặc biệt ta cần sử dụng cách viết sau (thêm dấu \):

Cách viết	Ký tự
'\''	'
'\"'	"
'\\'	\
'\n'	\n (chuyển dòng)
'\0'	\0 (null)
'\t'	Tab
'\b'	Backspace
'\r'	CR (về đầu dòng)
'\f'	LF (sang trang)

Chú ý:

Cần phân biệt hằng ký tự '0' và '\0'. Hằng '0' ứng với chữ số 0 có mã ASCII là 48, còn hằng '\0' ứng với kýtự \0 (thường gọi là ký tự null) có mã ASCII là 0.

Hằng ký tự thực sự là một số nguyên, vì vậy có thể dùng các số nguyên hệ 10 để biểu diễn các ký tự, ví dụ lệnh printf("%c%c", 65, 66) sẽ in ra AB.

- **Hằng xâu ký tự:** Hằng xâu ký tự là một dãy ký tự bất kỳ đặt trong hai dấu nháy kép.

Ví dụ 2.7:

#define xau1 "Ha noi"

#define xau2 "My name is Giang"

Xâu ký tự được lưu trữ trong máy dưới dạng một bảng có các phần tử là các ký tự riêng biệt. Trình biên dịch tự động thêm ký tự null \0 vào cuối mỗi xâu (ký tự \0 được xem là dấu hiệu kết thúc của một xâu ký tự).

Chú ý: Cần phân biệt hai hằng 'a' và "a". 'a' là hằng ký tự được lưu trữ trong 1 byte, còn "a" là hằng xâu ký tự được lưu trữ trong 1 mảng hai phần tử: phần tử thứ nhất chứa chữ a còn phần tử thứ hai chứa \0.

2.1.5. Cấu trúc một chương trình C

Cấu trúc chương trình và hàm là một trong các vấn đề quan trọng của C.

Hàm là một đơn vị độc lập của chương trình. Tính độc lập của hàm thể hiện ở hai điểm:

- Không cho phép xây dựng một hàm bên trong các hàm khác.

- Mỗi hàm có các biến, mảng .. riêng của nó và chúng chỉ được sử dụng nội bộ bên trong hàm. Nói cách khác hàm là đơn vị có tính chất khép kín.

Một chương trình bao gồm một hoặc nhiều hàm. Hàm int main() là thành phần bắt buộc của chương trình. Chương trình bắt đầu thực hiện các câu lệnh đầu tiên của hàm int main() và kết thúc khi gặp dấu } cuối cùng của hàm này. Khi chương trình làm việc, máy có thể chạy từ hàm này sang hàm khác.

Các chương trình C được tổ chức theo mẫu:

```
.....
hàm 1
.....
hàm 2
.....
```

```
.....
hàm n
```

Bên ngoài các hàm ở các vị trí (.....) là chỗ đặt: các toán tử #include ... (dùng để khai báo sử dụng các hàm chuẩn), toán tử #define ... (dùng để định nghĩa các hằng), định nghĩa kiểu dữ liệu bằng typedef, khai báo các biến ngoài, mảng ngoài....

Việc truyền dữ liệu và kết quả từ hàm này sang hàm khác được thực hiện theo một trong hai cách:

- Sử dụng đối của hàm.

- Sử dụng biến ngoài, mảng ngoài ...

Vậy nói tóm lại cấu truc cơ bản của chương trình như sau:

- Các #include

- Các #define

- Khai báo các đối tượng dữ liệu ngoài (biến, mảng, cấu trúc vv..).

- Khai báo nguyên mẫu các hàm.

- Hàm int main().

- Định nghĩa các hàm (hàm main có thể đặt sau hoặc xen vào giữa các hàm khác).

Ví dụ 2.8: Chương trình tính x lũy thừa y rồi in ra màn hình kết quả:

```
/* Chương trình Hello */
#include<stdio.h>
int main()
{
     printf("Hello World");
}
```

Một số qui tắc cần nhớ khi viết chương trình:

- Qui tắc đầu tiên cần nhớ là: Mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải kết thúc bằng dấu ;

- Qui tắc thứ hai là: C*ác lời giải thích cần được đặt giữa các dấu /* và */ và có thể được viết*

Trên một dòng

Trên nhiều dòng

Trên phần còn lại của dòng

Qui tắc thứ ba là: Trong chương trình, khi ta sử dụng các hàm chuẩn, ví dụ như printf(), getch(), ... mà các hàm này lại chứa trong file stdio.h trong thư mục của C, vì vậy ở đầu chương trình ta phải khai báo sử dụng ;

#include <stdio.h>

- Qui tắc thứ tư là: Một chương trình có thể chỉ có một hàm chính (hàm int main()) hoặc có thể có thêm vài hàm khác.

2.2. Biểu thức và các phép toán trong C

2.2.1. Biểu thức

Toán hạng có thể xem là một đại lượng có một giá trị nào đó. Toán hạng bao gồm hằng, biến, phần tử mảng và hàm.

Biểu thức lập nên từ các toán hạng và các phép tính để tạo nên những giá trị mới. Biểu thức dùng để diễn đạt một công thức, một qui trình tính toán, vì vậy nó là một thành phần không thể thiếu trong chương trình.

Biểu thức là một sự kết hợp giữa các phép toán và các toán hạng để diễn đạt một công thức toán học nào đó. Mỗi biểu thức có sẽ có một giá trị. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là biểu thức.

Trong C, ta có hai khái niệm về biểu thức:

- Biểu thức gán

- Biểu thức điều kiện

Biểu thức được phân loại theo kiểu giá trị: nguyên và thực. Trong các mệnh đề logic, biểu thức được phân thành đúng (giá trị khác 0) và sai (giá trị bằng 0).

Biểu thức thường được dùng trong:

- Vế phải của câu lệnh gán.

- Làm tham số thực sự của hàm.

- Làm chỉ số.

- Trong các toán tử của các cấu trúc điều khiển.

Tới đây, ta đã có hai khái niệm chính tạo nên biểu thức đó là toán hạng và phép toán. Toán hạng gồm: hằng, biến, phần tử mảng và hàm trước đây ta đã xét. Dưới đây ta sẽ nói đến các phép toán.

Lệnh gán và biểu thức:

Biểu thức gán là biểu thức có dạng:

v = e

Trong đó v là một biến (hay phần tử mảng), e là một biểu thức. Giá trị của biểu thức gán là giá trị của e, kiểu của nó là kiểu của v. Nếu đặt dấu ; vào sau biểu thức gán ta sẽ thu được phép toán gán có dạng:

v = e;

Biểu thức gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác. Ví dụ như khi ta viết

a = b = 5;

thì điều đó có nghĩa là gán giá trị của biểu thức b = 5 cho biến a. Kết qủa là b = 5 và a = 5.

Hoàn toàn tương tự như:

a = b = c = d = 6; gán 6 cho cả a, b, c và d

Ví dụ 2.9:

z = (y = 2) * (x = 6); //Ở đây * là phép toán nhân

gán 2 cho y, 6 cho x và nhân hai biểu thức lại cho ta z = 12.

2.2.2. Các phép toán

a) Các phép toán số học

Các phép toán hai ngôi số học là

Phép toán	ý nghĩa	Ví dụ
+	Phép cộng	a+b
-	Phép trừ	a-b
*	Phép nhân	a * b
/	Phép chia	a/b (Chia số nguyên sẽ chặt phần thập phân)
%	Phép lấy phần dư	a%b (Cho phần dư của phép chia a cho b)

Có phép toán một ngôi - ví du -(a+b) sẽ đảo giá trị của phép cộng (a+b).

Ví dụ 2.10:

11/3 = 3

11%3 = 2

-(2+6) = -8

Các phép toán + và - có cùng thứ tự ưu tiên, có thứ tự ưu tiên nhỏ hơn các phép * , / , % và cả ba phép này lại có thứ tự ưu tiên nhỏ hơn phép trừ một ngôi.

Các phép toán số học được thực hiện từ trái sang phải. Số ưu tiên và khả năng kết hợp của phép toán được chỉ ra trong một mục sau này

b) Các phép toán quan hệ và logic

Phép toán quan hệ và logic cho ta giá trị đúng (1) hoặc giá trị sai (0). Nói cách khác, khi các điều kiện nêu ra là đúng thì ta nhận được giá trị 1, ngược lại ta nhận giá trị 0.

Các phép toán quan hệ là:

Phép toán	ý nghiã	Ví dụ
>	So sánh lớn hơn	a>b 4>5 có giá trị 0
>=	So sánh lớn hơn hoặc bằng	a >= b 6 >= 2 có giá trị 1
<	So sánh nhỏ hơn	a<b 6<= 7 có giá trị 1
<=	So sánh nhỏ hơn hoặc bằng	a<= b 8<= 5 có giá trị 0
==	So sánh bằng nhau	a == b 6 == 6 có giá trị 1
!=	So sánh khác nhau	a!= b 9!= 9 có giá trị 0

Bốn phép toán đầu có cùng số ưu tiên, hai phép sau có cùng số thứ tự ưu tiên nhưng thấp hơn số thứ tự của bốn phép đầu.

Các phép toán quan hệ có số thứ tự ưu tiên thấp hơn so với các phép toán số học, cho nên biểu thức: i<n-1 được hiểu là i<(n-1).

Các phép toán logic:

Phép phủ định một ngôi !

a	**!a**
khác 0	0
bằng 0	1

Phép và (AND) &&, phép hoặc (OR) ||

a	b	a&&b	a||b
khác 0	khác 0	1	1
khác 0	bằng 0	0	1
bằng 0	khác 0	0	1

bằng 0	bằng 0	0	0

Các phép quan hệ có số ưu tiên nhỏ hơn so với ! nhưng lớn hơn so với && và ||, vì vậy biểu thức như: (a<b) && (c>d) có thể viết lại thành: a<b && c>d

Chú ý: Cả a và b có thể là nguyên hoặc thực.

c) Phép toán tăng giảm

C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và thực). Toán tử tăng là ++ sẽ cộng 1 vào toán hạng của nó, toán tử giảm -- thì sẽ trừ toán hạng đi 1.

Ví dụ 2.11:

```
n = 5
++n     //Cho ta n = 6
--n     //Cho ta n = 4
```

Ta có thể viết phép toán ++ và -- trước hoặc sau toán hạng như sau: ++n, n++, --n, n--.

Sự khác nhau của ++n và n++ ở chỗ: trong phép n++ thì tăng sau khi giá trị của nó đã được sử dụng, còn trong phép ++n thì n được tăng trước khi sử dụng. Sự khác nhau giữa n-- và --n cũng như vậy.

Ví dụ 2.12:

```
n = 5
x = ++n         //Cho ta x = 6 và n = 6
x = n++         //Cho ta x = 5 và n = 6
```

d) Thứ tự ưu tiên các phép toán:

Các phép toán có độ ưu tiên khác nhau, điều này có ý nghĩa trong cùng một biểu thức sẽ có một số phép toán này được thực hiện trước một số phép toán khác.

Thứ tự ưu tiên của các phép toán được trình bày trong bảng sau:

TT	**Phép toán**	**Trình tự kết hợp**
1	() [] ->	Trái qua phải

2	! ~ & * - ++ -- (type) sizeof	Phải qua trái
3	* (phép nhân) / %	Trái qua phải
4	+ -	Trái qua phải
5	<< >>	Trái qua phải
6	< <= > >=	Trái qua phải
7	== !=	Trái qua phải
8	&	Trái qua phải
9	^	Trái qua phải
10	|	Trái qua phải
11	&&	Trái qua phải
12	||	Trái qua phải
13	?:	Phải qua trái
14	= + = - = * = / = % = <<= > >= & = ^ = | =	Phải qua trái
15	,	Trái qua phải

Chú thích:

- Các phép toán tên một dòng có cùng thứ tự ưu tiên, các phép toán ở hàng trên có số ưu tiên cao hơn các số ở hàng dưới.

- Đối với các phép toán cùng mức ưu tiên thì trình tự tính toán có thể từ trái qua phải hay ngược lại được chỉ ra trong cột *trình tự kết hợp*.

Ví dụ 2.13:

*--px = *(--px) (Phải qua trái)

8/4*6 = (8/4)*6 (Trái qua phải)

Nên dùng các dấu ngoặc tròn để viết biểu thức một cách chính xác.

Các phép toán lạ:

Dòng 1:

- [ ] Dùng để biểu diễn phần tử mảng, ví dụ: a[i][j]

- . Dùng để biểu diễn thành phần cấu trúc, ví dụ: ht.ten

- -> Dùng để biểu diễn thành phần cấu trúc thông qua con trỏ

Dòng 2:

- * Dùng để khai báo con trỏ, ví dụ: int *a

- & Phép toán lấy địa chỉ, ví dụ: &x

- (type) là phép chuyển đổi kiểu, ví dụ: (float)(x+y)

Dòng 15: Toán tử **,** thường dùng để viết một dãy biểu thức trong toán tử for.

e) Chuyển đổi kiểu giá trị

Việc chuyển đổi kiểu giá trị thường diễn ra một cách tự động trong hai trường hợp sau:

Khi gán biểu thức gồm các toán hạng khác kiểu.

Khi gán một giá trị kiểu này cho một biến (hoặc phần tử mảng) kiểu khác. Điều này xảy ra trong toán tử gán, trong việc truyền giá trị các tham số thực sự cho các đối.

Ngoài ra, ta có thể chuyển từ một kiểu giá trị sang một kiểu bất kỳ mà ta muốn bằng phép chuyển sau:

(type) biểu thức

Ví dụ 2.14:

(float) (a+b)

Chuyển đổi kiểu trong biểu thức:

Khi hai toán hạng trong một phép toán có kiểu khác nhau thì kiểu thấp hơn sẽ được nâng thành kiểu cao hơn trước khi thực hiện phép toán. Kết quả thu được là một giá trị kiểu cao hơn. Chẳng hạn:

- Giữa int và long thì int chuyển thành long.

- Giữa int và float thì int chuyển thành float.

- Giữa float và double thì float chuyển thành double.

Ví dụ 2.15:

1.5*(11/3) = 4.5

1.5*11/3 = 5.5

(11/3)*1.5 = 4.5

Chuyển đổi kiểu thông qua phép gán:

Giá trị của vế phải được chuyển sang kiểu vế trái đó là kiểu của kết quả. Kiểu int có thể được được chuyển thành float. Kiểu float có thể chuyển thành int do chặt đi phần thập phân. Kiểu double chuyển thành float bằng cách làm tròn. Kiểu long được chuyển thành int bằng cách cắt bỏ một vài chữ số.

Ví dụ 2.16:

int n;

n = 15.6 giá trị của n là 15

Đổi kiểu dạng (type)biểu thức:

Theo cách này, kiểu của biểu thức được đổi thành kiểu type theo nguyên tắc trên.

Ví dụ 2.17:

Phép toán: (int)a

Cho một giá trị kiểu int. Nếu a là float thì ở đây có sự chuyển đổi từ float sang int. Chú ý rằng bản thân kiểu của a vẫn không bị thay đổi. Nói cách khác, a vẫn có kiểu float nhưng (int)a có kiểu int.

Đối với hàm toán học của thư viện chuẩn, thì giá trị của đối và giá trị của hàm đều có kiểu double, vì vậy để tính căn bậc hai của một biến nguyên n ta phải dùng phép ép kiểu để chuyển kiểu int sang double như sau:

sqrt((double)n)

Phép ép kiểu có cùng số ưu tiên như các toán tử một ngôi.

Chú ý: Muốn có giá trị chính xác trong phép chia hai số nguyên cần dùng phép ép kiểu:

((float)a)/b

Để đổi giá trị thực r sang nguyên, ta dùng:

(int)(r+0.5)

Chú ý thứ tự ưu tiên:

(int)1.4*10 = 1*10 = 10

(int)(1.4*10) = (int)14.0 = 14

2.2.3. Biểu thức điều kiện

Toán tử điều kiện tính toán một biểu thức và trả về một giá trị khác tuỳ thuộc vào biểu thức đó là đúng hay sai.

Cú pháp:

<điều kiện>?<kết quả 1>: <kết quả 2>

Nếu <***Điều kiện***> là **true** thì giá trị trả về sẽ là ***kết quả 1***, nếu không giá trị trả về là ***kết quả 1***.

7==5?4: 3 //Trả về 3 vì 7 không bằng 5.

7==5+2?4: 3 //Trả về 4 vì 7 bằng 5+2.

5>3?a: b //Trả về a, vì 5 lớn hơn 3.

a>b?a: b //Trả về giá trị lớn hơn, a hoặc b.

2.3. Khai báo biến

2.3.1. Khai báo biến

Mỗi biến cần phải được khai báo trước khi đưa vào sử dụng. Việc khai báo biến được thực hiện theo mẫu sau:

Kiểu dữ liệu của biến tên biến:

Ví dụ 2.18:

int a, b, c	Khai báo ba biến int là a, b, c
long dai, mn	Khai báo hai biến long là dai và mn
char kt1, kt2	Khai báo hai biến ký tự là kt1 và kt2
float x, y	Khai báo hai biến float là x và y

double canh1, canh2	Khai báo hai biến double là canh1 và canh2

Biến kiểu int chỉ nhận được các giá trị kiểu int. Các biến khác cũng có ý nghĩa tương tự. Các biến kiểu char chỉ chứa được một ký tự. Để lưu trữ được một xâu ký tự cần sử dụng một mảng kiểu char.

Vị trí của khai báo biến:

Các khai báo cần phải được đặt ngay sau dấu { đầu tiên của thân hàm và cần đứng trước mọi câu lệnh khác. Sau đây là một ví dụ về khai báo biến sai:

(Khái niệm về hàm và cấu trúc chương trình sẽ nghiên cứu sau này)

```
int main()
{
     int a, b, c;
     a = 2;
     int d; /* Vị trí của khai báo sai */
     .....
}
```

Khởi đầu cho biến: Nếu trong khai báo ngay sau tên biến ta đặt dấu = và một giá trị nào đó thì đây chính là cách vừa khai báo vừa khởi đầu cho biến.

Ví dụ 2.19:

int a, b = 20, c, d = 40;

float e = -55.2, x = 27.23, y, z, t = 18.98;

Việc khởi đầu và việc khai báo biến rồi gán giá trị cho nó sau này là hoàn toàn tương đương.

Lấy địa chỉ của biến:

Mỗi biến được cấp phát một vùng nhớ gồm một số byte liên tiếp. Số hiệu của byte đầu chính là địa chỉ của biến. Địa chỉ của biến sẽ được sử dụng trong một số hàm ta sẽ nghiên cứu sau này (ví dụ như hàm scanf).

Để lấy địa chỉ của một biến ta sử dụng phép toán:

& tên biến

2.3.2. Phạm vi của biến

Khi lập trình, bạn phải nắm rõ phạm vi của biến. Nếu khai báo và sử dụng không đúng, không rõ ràng sẽ dẫn đến sai sót khó kiểm soát được, vì vậy bạn cần phải xác định đúng vị trí, phạm vi sử dụng biến trước khi sử dụng biến.

- Khai báo biến ngoài (biến toàn cục): Vị trí biến đặt bên ngoài tất cả các hàm, cấu trúc... Các biến này có ảnh hưởng đến toàn bộ chương trình. Chu trình sống của nó là bắt đầu chạy chương trình đến lúc kết thúc chương trình.

- Khai báo biến trong (biến cục bộ): Vị trí biến đặt bên trong hàm, cấu trúc…. Chỉ ảnh hưởng nội bộ bên trong hàm, cấu trúc đó…. Chu trình sống của nó bắt đầu từ lúc hàm, cấu trúc được gọi thực hiện đến lúc thực hiện xong.

2.4. Ghi chú

Trong khi lập trình cần phải ghi chú để giải thích các biến, hằng, thao tác xử lý giúp cho chương trình rõ ràng dễ hiểu, dễ nhớ, dễ sửa chữa và để người khác đọc vào dễ hiểu. Trong C có các ghi chú sau: // hoặc /* nội dung ghi chú */

Ví dụ 2.20:

```
#include <stdio.h>
int main()
{
     int a, b;      //khai bao bien t kieu int
     a = 1;         //gan 1 cho a
     b = 3;         //gan 3 cho b
     /*   thuat toan tim so lon nhat la
          neu a lon hon b thi a lon nhat
          nguoc lai b lon nhat */
     if (a > b)
          printf("max: %d", a);
     else
```

```
        printf("max: %d", b);
}
```

Khi biên dịch chương trình, C gặp cặp dấu ghi chú sẽ không dịch ra ngôn ngữ máy.

Tóm lại, đối với ghi chú dạng // dùng để ghi chú một hàng và dạng /* …. */ có thể ghi chú một hàng hoặc nhiều hàng.

2.5. Nhập / Xuất dữ liệu trong C

Phần này giới thiệu thư viện vào/ra chuẩn là một tập các hàm được thiết kế để cung cấp hệ thống vào/ra chuẩn cho các chương trình C. Chúng ta sẽ không mô tả toàn bộ thư viện vào ra ở đây mà chỉ quan tâm nhiều hơn đến việc nêu ra những điều cơ bản nhất để viết chương trình C tương tác với môi trường và hệ điều hành.

2.5.1. Hàm printf

Kết xuất dữ liệu được định dạng.

Cú pháp:

printf ("chuỗi định dạng"[, đối mục 1, đối mục 2, …]);

Khi sử dụng hàm phải khai báo tiền xử lý **#include <stdio.h>**

- printf: tên hàm, phải viết bằng chữ thường.

- đối mục 1, …: là các mục dữ kiện cần in ra màn hình. Các đối mục này có thể là biến, hằng hoặc biểu thức phải được định trị trước khi in ra.

- chuỗi định dạng: được đặt trong cặp nháy kép (" "), gồm 3 loại:

+ Đối với chuỗi kí tự ghi như thế nào in ra giống như vậy.

+ Đối với những kí tự chuyển đổi dạng thức cho phép kết xuất giá trị của các đối mục ra màn hình tạm gọi là mã định dạng. Sau đây là các dấu mô tả định dạng:

%c : Ký tự đơn

%s : Chuỗi

%d : Số nguyên thập phân có dấu

%f : Số chấm động (ký hiệu thập phân)

%e : Số chấm động (ký hiệu có số mũ)

%g : Số chấm động (%f hay %g)

%x : Số nguyên thập phân không dấu

%u : Số nguyên hex không dấu

%o : Số nguyên bát phân không dấu

l : Tiền tố dùng kèm với %d, %u, %x, %o để chỉ số nguyên dài (ví dụ %ld)

+ Các ký tự điều khiển và ký tự đặc biệt

\n : Nhảy xuống dòng kế tiếp canh về cột đầu tiên.

\t :Canh cột tab ngang.

\r : Nhảy về đầu hàng, không xuống hàng.

\a : Tiếng kêu bip.

\\ : In ra dấu \

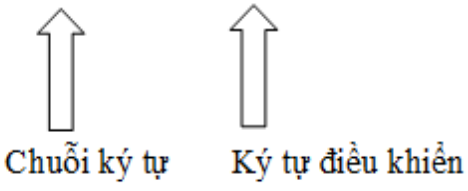
\" : In ra dấu "

\' : In ra dấu '

%%: In ra dấu %

Ví dụ 2.21:

```
printf("Bai hoc C dau tien. \n");
```

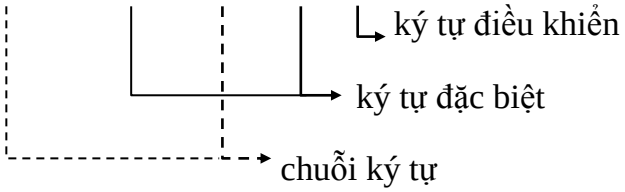


Kết quả in ra màn hình:

```
Bai hoc C dau tien.

_
```

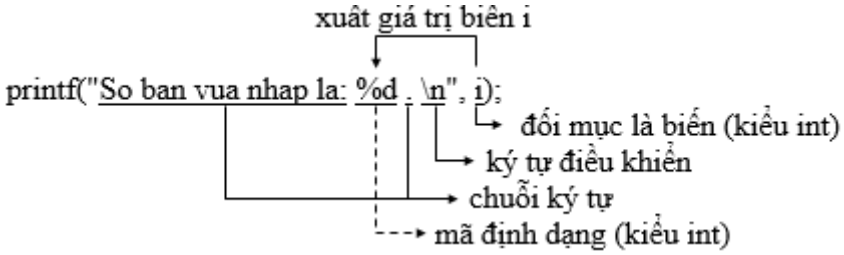
printf("Ma dinh dang \\\" in ra dau \" . \n");



Kết quả in ra màn hình

```
Ma dinh dang \" in ra dau "
_
```

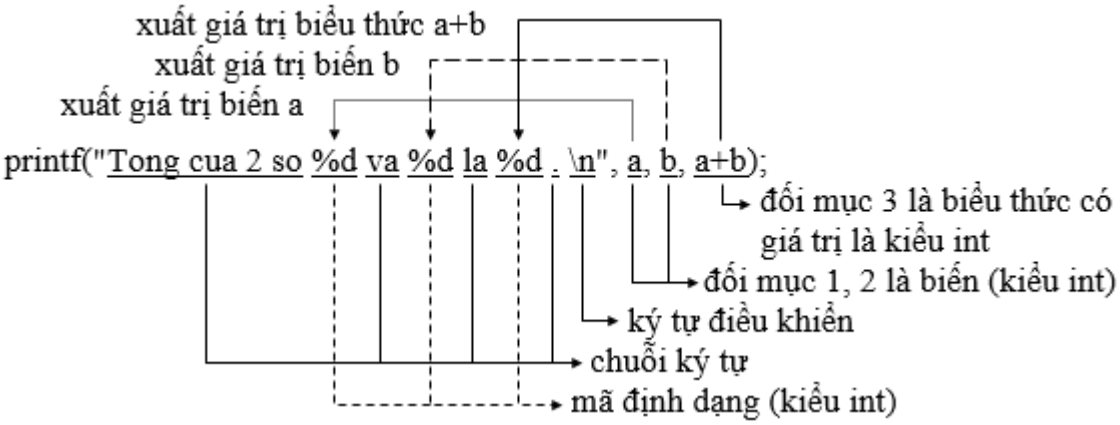
Giả sử biến i có giá trị = 5



Kết quả in ra màn hình

```
So ban vua nhap la: 5.
_
```

Giả sử biến a có giá trị = 7 và b có giá trị = 4



Kết quả in ra màn hình

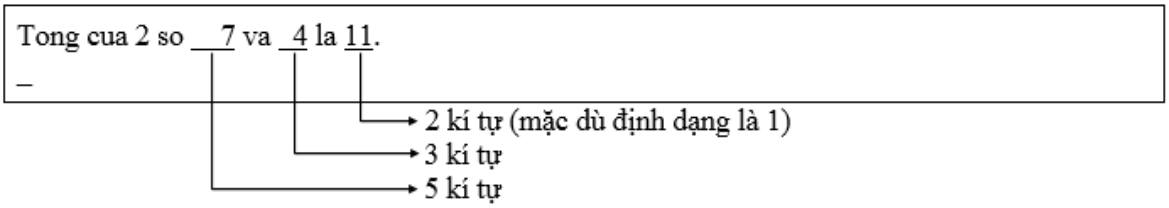
```
Tong cua 2 so 7 va 4 la 11.
_
```

Sửa lại Ví dụ

printf("Tong cua 2 so %5d va %3d la %1d . \n", a, b, a+b);

Bề rộng trường

Kết quả in ra màn hình



2.5.2. Hàm scanf

Cú pháp:

scanf ("chuỗi định dạng"[, đối mục 1, đối mục 2, …]);

Khi sử dụng hàm phải khai báo tiền xử lý **#include <stdio.h>**

- scanf: tên hàm, phải viết bằng chữ thường.

- khung định dạng: được đặt trong cặp nháy kép (" ") là hình ảnh dạng dữ liệu nhập vào.

- Đối mục 1, …: là danh sách các đối mục cách nhau bởi dấu phẩy, mỗi đối mục sẽ tiếp nhận giá trị nhập vào.

Ví dụ 2.22:

scanf("%d", &i);

đối mục 1

mã định dạng

Nhập vào 12abc, biến i chỉ nhận giá trị 12. Nhập 3.4 chỉ nhận giá trị 3.

scanf("%d%d", &a, &b);

Nhập vào 2 số a, b phải cách nhau bằng **khoảng trắng** hoặc **enter**.

scanf("%d/%d/%d", &ngay, &thang, &nam);

Nhập vào ngày, tháng, năm theo dạng ngay/thang/nam (20/12/2002)

Ví dụ 2.23: scanf("%d%*c%d%*c%d", &ngay, &thang, &nam);

Nhập vào ngày, tháng, năm với dấu phân cách /, -, …; ngoại trừ số.

scanf("%2d%2d%4d", &ngay, &thang, &nam);

Nhập vào ngày, tháng, năm theo dạng dd/mm/yyyy.