HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

BÀI GIẢNG

CÁC KỸ THUẬT LẬP TRÌNH

Hà Nội 2013

LỜI NÓI ĐẦU

Sự phát triển công nghệ thông tin trong những năm vừa qua đã làm thay đổi bộ mặt kinh tế xã hội toàn cầu, trong đó công nghệ phần mềm trở thành một ngành công nghiệp quan trọng đầy tiềm năng. Với sự hội tụ của công nghệ viễn thông và công nghệ thông tin, tỷ trọng về giá trị phần mềm chiếm rất cao trong các hệ thống viễn thông cũng như các thiết bị đầu cuối. Chính vì lý do đó, việc nghiên cứu, tìm hiểu, tiến tới phát triển cũng như làm chủ các hệ thống phần mềm của các kỹ sư điện tử viễn thông là rất cần thiết.

Môn học *Kỹ thuật lập trình* là môn học cơ sở bắt buộc đối với sinh viên chuyên ngành điện tử viễn thông và công nghệ thông tin của Học viện công nghệ Bưu chính Viễn thông.

Cuốn giáo trình *"Kỹ thuật lập trình"*, được hình thành trên cơ sở các kinh nghiệm đã được đúc rút từ bài giảng của môn học Kỹ thuật lập trình cho sinh viên các ngành nói trên trong những năm học vừa qua với mục đích cung cấp cho sinh viên những kiến thức cơ bản nhất, có tính hệ thống liên quan tới môn học này.

Khi học môn *Kỹ thuật lập trình*, sinh viên chỉ cần học qua môn "Tin học cơ sở" và chỉ cần thế các bạn đã có đủ kiến thức cơ sở cần thiết để tiếp thu kiến thức của *Kỹ thuật lập trình*.

Thông qua cuốn giáo trình này, chúng tôi muốn giới thiệu với các bạn đọc về kỹ năng lập trình cấu trúc thông qua một số thuật toán quan trọng, bao gồm: Đại cương về lập trình cấu trúc; Con trỏ và mảng; Duyệt và đệ qui; Ngăn xếp, hàng đợi và danh sách móc nối; Cây; Đồ thị và cuối cùng là Sắp xếp và tìm kiếm. Phần phụ lục là bài tập tổng hợp lại những kiến thức cơ bản nhất đã được đề cập trong giáo trình và được thể hiện bằng một chương trình.

Tuy đã rất chú ý và cẩn trọng trong quá trình biên soạn, nhưng giáo trình chắc chắn không tránh khỏi những thiếu sót và hạn chế. Chúng tôi xin chân thành mong bạn đọc đóng góp ý kiến để giáo trình nay ngày càng hoàn thiện hơn. Mọi sự đóng góp ý kiến xin gửi về Khoa Công nghệ thông tin – Học viện Công nghệ Bưu chính Viễn thông.

Chúng tôi xin tỏ lòng biết ơn tới TS. Từ Minh Phương, giảng viên khoa Công nghệ thông tin – Học viện Công nghệ Bưu chính Viễn thông đã đọc và hiệu đính lại toàn bộ bản thảo của giáo trình này.

Hà Nội, ngày 24 tháng 8 năm 2002

Các tác giả

MỤC LỤC

CHƯƠNG 1. MỞ ĐẦU

1.1. Sơ lược về lịch sử lập trình cấu trúc

Lập trình là một trong những công việc nặng nhọc nhất của khoa học máy tính. Có thể nói, năng suất xây dựng các sản phẩm phần mềm là rất thấp so với các hoạt động trí tuệ khác. Một sản phẩm phần mềm có thể được thiết kế và cài đặt trong vòng 6 tháng với 3 lao động chính. Nhưng để kiểm tra tìm lỗi và tiếp tục hoàn thiện sản phẩm đó phải mất thêm chừng 3 năm. Đây là hiện tượng phổ biến trong tin học của những năm 1960 khi xây dựng các sản phẩm phần mềm bằng kỹ thuật lập trình tuyến tính. Để khắc phục tình trạng lỗi của sản phẩm, người ta che chắn nó bởi một mành che mang tính chất thương mại được gọi là Version. Thực chất, Version là việc thay thế sản phẩm cũ bằng cách sửa đổi nó rồi công bố dưới dạng một Version mới, giống như: MS-DOS 4.0 chỉ tồn tại trong thời gian vài tháng rồi thay đổi thành MS-DOS 5.0, MS-DOS 5.5, MS-DOS 6.0 . . . Đây không phải là một sản phẩm mới như ta tưởng mà trong nó còn tồn tại những lỗi không thể bỏ qua được, vì ngay MS-DOS 6.0 cũng chỉ là sự khắc phục hạn chế của MS-DOS 3.3 ban đầu.

Trong thời kỳ đầu của tin học, các lập trình viên xây dựng chương trình bằng các ngôn ngữ lập trình bậc thấp, quá trình nạp và theo dõi hoạt động của chương trình một cách trực tiếp trong chế độ trực tuyến (on-line). Việc tìm và sửa lỗi (debbugging) như ngày nay là không thể thực hiện được. Do vậy, trước những năm 1960, người ta coi việc lập trình giống như những hoạt động nghệ thuật nhuộm màu sắc cá nhân hơn là khoa học. Một số người nắm được một vài ngôn ngữ lập trình, cùng một số mẹo vặt tận dụng cấu hình vật lý cụ thể của hệ thống máy tính, tạo nên một số món lạ của phần mềm được coi là một chuyên gia nắm bắt được những bí ẩn của nghệ thuật lập trình.

Các hệ thống máy tính trong giai đoạn này có cấu hình yếu, bộ nhớ nhỏ, tốc độ các thiết bị vào ra thấp làm chậm quá trình nạp và thực hiện chương trình. Chương trình được xây dựng bằng kỹ thuật lập trình tuyến tính mà nổi bật nhất là ngôn ngữ lập trình Assembler và Fortran. Với phương pháp lập trình tuyến tính, lập trình viên chỉ được phép thể hiện chương trình của mình trên hai cấu trúc lệnh, đó là cấu trúc lệnh tuần tự (sequential) và nhảy không điều kiện (goto). Hệ thống thư viện vào ra nghèo nàn làm cho việc lập trình trở nên khó khăn, chi phí cho các sản phẩm phần mềm quá lớn, độ tin cậy của các sản phẩm phần mềm không cao dẫn tới hàng loạt các dự án tin học bị thất bại, đặc biệt là các hệ thống tin học có tầm cỡ lớn. Năm 1973, Hoare khẳng định, nguyên nhân thất bại mà người Mỹ gặp phải khi phóng vệ tinh nhân tạo về phía sao Vệ nữ (Sao Kim) là do lỗi của chương trình điều khiển viết bằng Fortran. Thay vì viết:

DO 50 I = 12, 523

(thực hiện số 50 với I là 12, 13, ..., 523)

Lập trình viên (hoặc thao tác viên đục bìa) viết thành:

DO 50 I = 12.523

(Dấu phảy đã thay bằng dấu chấm)

Gặp câu lệnh này, chương trình dịch của Fortran đã hiểu là gán giá trị thực 12.523 cho biến DO50I làm cho kết quả chương trình sai.

Để giải quyết những vướng mắc trong kỹ thuật lập trình, các nhà tin học lý thuyết đã đi sâu vào nghiên cứu tìm hiểu bản chất của ngôn ngữ, thuật toán và hoạt động lập trình, nâng nội dung của kỹ thuật lập trình lên thành các nguyên lý khoa học ngày nay. Kết quả nổi bật nhất trong giai đoạn này là Knuth xuất bản bộ 3 tập sách mang tên "Nghệ thuật lập trình" giới thiệu hết sức tỉ mỉ cơ sở lý thuyết đảm bảo toán học và các thuật toán cơ bản xử lý dữ liệu nửa số, sắp xếp và tìm kiếm. Năm 1968, Dijkstra công bố lá thư " Về sự nguy hại của toán tử goto". Trong công trình này, Dijkstra khẳng định, có một số lỗi do goto gây nên không thể xác định được điểm bắt đầu của lỗi. Dijkstra còn khẳng định thêm: "Tay nghề của một lập trình viên tỉ lệ nghịch với số lượng toán tử goto mà anh ta sử dụng trong chương trình", đồng thời kêu gọi huỷ bỏ triệt để toán tử goto trong mọi ngôn ngữ lập trình ngoại trừ ngôn ngữ lập trình bậc thấp. Dijkstra còn đưa ra khẳng định, động thái của chương trình có thể được đánh giá tường minh qua các cấu trúc lặp, rẽ nhánh, gọi đệ qui là cơ sở của lập trình cấu trúc ngày nay.

Những kết quả được Dijikstra công bố đã tạo nên một cuộc cách mạng trong kỹ thuật lập trình, Knuth liệt kê một số trường hợp có lợi của goto như vòng lặp kết thúc giữa chừng, bắt lỗi . . ., Dijkstra, Hoare, Knuth tiếp tục phát triển tư tưởng coi chương trình máy tính cùng với lập trình viên là đối tượng nghiên cứu của kỹ thuật lập trình và phương pháp làm chủ sự phức tạp của các hoạt động lập trình. Năm 1969, Hoare đã phát biểu các tiên đề phục vụ cho việc chứng minh tính đúng đắn của chương trình, phát hiện tính bất biến của vòng lặp bằng cách coi chương trình vừa là bản mã hoá thuật toán đồng thời là bản chứng minh tính đúng đắn của chương trình. Sau đó Dahl, Hoare, Dijiksta đã phát triển thành ngôn ngữ lập trình cấu trúc.

Để triển khai các nguyên lý lập trình cấu trúc, L. Wirth đã thiết kế và cài đặt ngôn ngữ ALGOL W là một biến thể của ALGOL 60. Sau này, L. Wirth tiếp tục hoàn thiện để trở thành ngôn ngữ lập trình Pascal. Đây là ngôn ngữ lập trình giản dị, sáng sủa về cú pháp, dễ minh họa những vấn đề phức tạp của lập trình hiện đại và được coi là một chuẩn mực trong giảng dạy lập trình.

Năm 1978, Brian Barninghan cùng Denit Ritche thiết kế ngôn ngữ lập trình C với tối thiểu các cấu trúc lệnh và hàm khá phù hợp với tư duy và tâm lý của của người lập trình. Đồng thời, hai tác giả đã phát hành phiên bản hệ điều hành UNIX viết chủ yếu bằng ngôn ngữ C, khẳng định thêm uy thế của C trong lập trình hệ thống.

1.2. Cấu trúc lệnh - Lệnh có cấu trúc- Cấu trúc dữ liệu

1.2.1. Cấu trúc lệnh (cấu trúc điều khiển)

Mỗi chương trình máy tính về bản chất là một bản mã hoá thuật toán. Thuật toán được coi là dãy hữu hạn các thao tác sơ cấp trên tập đối tượng vào (Input) nhằm thu được kết quả ra (output). Các thao tác trong một ngôn ngữ lập trình cụ thể được điều khiển bởi các lệnh hay các cấu trúc điều khiển, còn các đối tượng chịu thao tác thì được mô tả và biểu diễn thông qua các cấu trúc dữ liệu.

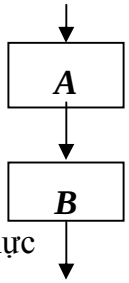
Trong các ngôn ngữ lập trình cấu trúc, những cấu trúc lệnh sau được sử dụng để xây dựng chương trình. Dĩ nhiên, chúng ta sẽ không bàn tới cấu trúc nhảy không điều kiện goto mặc dù ngôn ngữ lập trình cấu trúc nào cũng trang bị cấu trúc lệnh goto.

Cấu trúc tuần tự

A;

B;

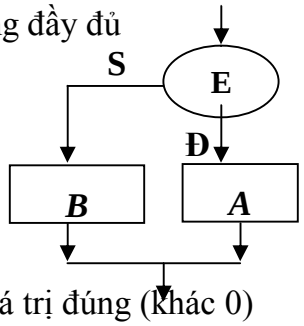
Sau khi thực hiện lệnh A thì thực hiện lệnh B



Cấu trúc rẽ nhánh dạng đầy đủ

If (E) A;

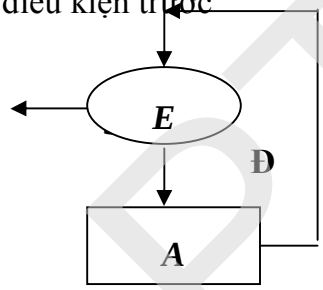
Else B;



Nếu biểu thức E có giá trị đúng (khác 0) thì thực hiện A; Nếu E sai thì thực hiện B;

Cấu trúc lặp với điều kiện trước

While (E) A;



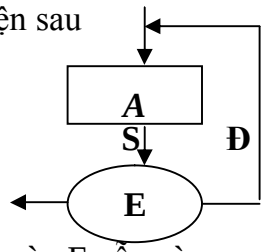
Trong khi biểu thức E còn có giá trị đúng thì thực hiện A;

Cấu trúc lặp với điều kiện sau

do

A;

while (E);

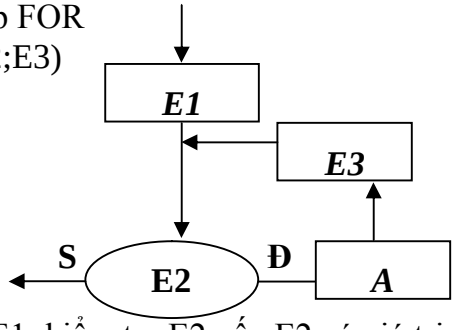


Thực hiện A cho tới khi nào E vẫn còn đúng;

Cấu trúc lặp FOR

For (E1; E2;E3)

A;



Thực hiện E1; kiểm tra E2 nếu E2 có giá trị đúng thì thực hiện A; Quá trình được lặp lại bằng việc thực hiện E3 và kiểm tra E2;

A, B : ký hiệu cho các câu lệnh đơn hoặc lệnh hợp thành. Mỗi lệnh đơn lẻ được gọi là một lệnh đơn, lệnh hợp thành là lệnh hay cấu trúc lệnh được ghép lại với nhau theo qui định của ngôn ngữ, trong Pascal là tập lệnh hay cấu trúc lệnh được bao trong thân của begin . . . end; trong C là tập các lệnh hay cấu trúc lệnh được bao trong hai ký hiệu { ... }.

E, E1, E2, E3 là các biểu thức số học hoặc logic. Một số ngôn ngữ lập trình coi giá trị của biểu thức logic hoặc đúng (TRUE) hoặc sai (FALSE), một số ngôn ngữ lập trình khác như C coi giá trị của biểu thức logic là đúng nếu nó có giá trị khác 0, ngược lại biểu thức logic có giá trị sai.

Cần lưu ý rằng, một chương trình được thể hiện bằng các cấu trúc điều khiển lệnh : tuần tự, tuyển chọn if..else, switch . . case .. default, lặp với điều kiện trước while , lặp với điều kiện sau do . . while, vòng lặp for bao giờ cũng chuyển được về một chương trình, chỉ sử dụng tối thiểu hai cấu trúc lệnh là tuần tự và lặp với điều kiện trước while. Phương pháp lập trình này còn được gọi là phương pháp lập trình hạn chế.

1.2.2. Lệnh có cấu trúc

Lệnh có cấu trúc là lệnh cho phép chứa các cấu trúc điều khiển trong nó. Khi tìm hiểu một cấu trúc điều khiển cần xác định rõ vị trí được phép đặt một cấu trúc điều khiển trong nó, cũng như nó là một phần của cấu trúc điều khiển nào. Điều này tưởng như rất tầm thường nhưng có ý nghĩa hết sức quan trọng trong khi xây dựng và kiểm tra lỗi có thể xảy ra trong chương trình. Nguyên tắc viết chương trình theo cấu trúc: Cấu trúc con phải được viết lọt trong cấu trúc cha, điểm vào và điểm ra của mỗi cấu trúc phải nằm trên cùng một hàng dọc. Ví dụ sau sẽ minh họa cho nguyên tắc viết chương trình:

```
if (E)
        while (E1)
                A;
else
        do
                B;
        while(E2);
```

Trong ví dụ trên, while (E1) A; là cấu trúc con nằm trong thân của cấu trúc cha là if (E) ; còn do B while(E2); là cấu trúc con trong thân của else. Do vậy, câu lệnh while(E1); do . . . while(E2) có cùng cấp với nhau nên nó phải nằm trên cùng một cột, tương tự như vậy với A, B và if với else.

1.2.3. Cấu trúc dữ liệu

Các ngôn ngữ lập trình cấu trúc nói chung đều giống nhau về cấu trúc lệnh và cấu trúc dữ liệu. Điểm khác nhau duy nhất giữa các ngôn ngữ lập trình cấu trúc là phương pháp đặt tên, cách khai báo, cú pháp câu lệnh và tập các phép toán được phép thực hiện trên các cấu trúc dữ liệu cụ thể. Nắm bắt được nguyên tắc này, chúng ta sẽ dễ dàng chuyển đổi cách thể hiện chương trình từ ngôn ngữ lập trình này sang ngôn ngữ lập trình khác một cánh nhanh chóng mà không tốn quá nhiều thời gian cho việc học tập ngôn ngữ lập trình.

Thông thường, các cấu trúc dữ liệu được phân thành hai loại: cấu trúc dữ liệu có kiểu cơ bản (Base type) và cấu trúc dữ liệu có kiểu do người dùng định nghĩa (User type) hay còn gọi là kiểu dữ liệu có cấu trúc. Kiểu dữ liệu cơ bản bao gồm: Kiểu kí tự (char), kiểu số nguyên có dấu (signed int), kiểu số nguyên không dấu (unsigned int), kiểu số nguyên dài có dấu (signed long), kiểu số nguyên dài không dấu (unsigned long), kiểu số thực (float) và kiểu số thực có độ chính xác gấp đôi (double).

Kiểu dữ liệu do người dùng định nghĩa bao gồm kiểu xâu kí tự (string), kiểu mảng (array), kiểu tập hợp (union), kiểu cấu trúc (struct), kiểu file, kiểu con trỏ (pointer) và các kiểu dữ liệu được định nghĩa mới hoàn toàn như kiểu danh sách móc nối (link list), kiểu cây (tree) . . .

Kích cỡ của kiểu cơ bản đồng nghĩa với miền xác định của kiểu với biểu diễn nhị phân của nó, và phụ thuộc vào từng hệ thống máy tính cụ thể. Để xác định kích cỡ của kiểu nên dùng toán tử sizeof(type). Chương trình sau sẽ liệt kê kích cỡ của các kiểu cơ bản.

Ví dụ 1.1. kiểm tra kích cỡ của kiểu.

```
#include      <stdio.h>
#include      <conio.h>
void main(void) {
      printf("\n Kích cỡ kiểu kí tự:%d", sizeof(char));
      printf("\n Kích cỡ kiểu kí tự không dấu:%d", sizeof(unsigned char));
      printf("\n Kích cỡ kiểu số nguyên không dấu:%d", sizeof(unsigned int));
      printf("\n Kích cỡ kiểu số nguyên có dấu:%d", sizeof(signed int));
      printf("\n Kích cỡ kiểu số nguyên dài không dấu:%d", sizeof(unsigned long ));
      printf("\n Kích cỡ kiểu số nguyên dài có dấu:%d", sizeof(signed long ));
      printf("\n Kích cỡ kiểu số thực có độ chính xác đơn:%d", sizeof(float ));
      printf("\n Kích cỡ kiểu số thực có độ chính xác kép:%d", sizeof(double ));
      getch();
}
```

Kích cỡ của các kiểu dữ liệu do người dùng định nghĩa là tổng kích cỡ của mỗi kiểu thành viên trong nó. Chúng ta cũng vẫn dùng toán tử sizeof(tên kiểu) để xác định độ lớn tính theo byte của các kiểu dữ liệu này.

Một điểm đặc biệt chú ý trong khi lập trình trên các cấu trúc dữ liệu là cấu trúc dữ liệu nào thì phải kèm theo phép toán đó, vì một biến được gọi là thuộc kiểu dữ liệu nào đó

nếu như nó nhận một giá trị từ miền xác định của kiểu và các phép toán trên kiểu dữ liệu đó.

1.3. Nguyên lý tối thiểu

Hãy bắt đầu từ một tập nguyên tắc và tối thiểu các phương tiện là các cấu trúc lệnh, kiểu dữ liệu cùng các phép toán trên nó và thực hiện viết chương trình. Sau khi nắm chắc những công cụ vòng đầu mới đặt vấn đề mở rộng sang hệ thống thư viện tiện ích của ngôn ngữ.

Khi làm quen với một ngôn ngữ lập trình nào đó, không nhất thiết phải lệ thuộc quá nhiều vào hệ thống thư viện hàm của ngôn ngữ, mà điều quan trọng hơn là trước một bài toán cụ thể, chúng ta sử dụng ngôn ngữ để giải quyết nó thế nào, và phương án tốt nhất là lập trình bằng chính hệ thống thư viện hàm của riêng mình. Do vậy, đối với các ngôn ngữ lập trình, chúng ta chỉ cần nắm vững một số các công cụ tối thiểu như sau:

1.3.1. Tập các phép toán

Tập các phép toán số học: + (cộng); - (trừ); * (nhân); % (lấy phần dư); / (chia).

Tập các phép toán số học mở rộng:

++a ⇔ a = a +1; // tăng giá trị biến nguyên a lên một đơn vị;

--a ⇔ a = a-1; //giảm giá trị biến nguyên a một đơn vị;

a+= n ⇔ a = a+n; // tăng giá trị biến nguyên a lên n đơn vị;

a-=n ⇔ a = a - n; // giảm giá trị biến nguyên a n đơn vị);

a%=n ⇔ a = a%n; // lấy giá trị biến a modul với n;

a/=n ⇔a=a/n;// lấy giá trị biến a chia cho n;

a*=n ⇔ a = a*n; // lấy giá trị biến a nhân với n;

Tập các phép toán so sánh: >, <, >=, <=, ==, != (lớn hơn, nhỏ hơn, lớn hơn hoặc bằng, nhỏ hơn hoặc bằng, đúng bằng, khác). Qui tắc viết được thể hiện như sau:

```
if ( a>b) { . . } // nếu a lớn hơn b
if ( a<b) { . . } // nếu a nhỏ hơn b
if ( a>=b) { . . } // nếu a lớn hơn hoặc bằng b
if ( a<=b) { . . } // nếu a nhỏ hơn hoặc bằng b
if ( a==b) { . . } // nếu a đúng bằng  b
if ( a!=b) { . . } // nếu a khác  b
```

Tập các phép toán logic: &&, ||, ! (và, hoặc, phủ định)

&& : Phép và logic chỉ cho giá trị đúng khi hai biểu thức tham gia đều có giá trị đúng (giá trị đúng của một biểu thức trong C được hiểu là biểu thức có giá trị khác 0).
|| : Phép hoặc logic chỉ cho giá trị sai khi cả hai biểu thức tham gia đều có giá trị sai.

! : Phép phủ định cho giá trị đúng nếu biểu thức có giá trị sai và ngược lại cho giá trị sai khi biểu thức có giá trị đúng. Ngữ nghĩa của các phép toán được minh họa thông qua các câu lệnh sau:

```
int a =3, b =5;
    if ( (a !=0) && (b!=0) )  // nếu a khác 0 và b khác 0
    if ((a!=0) || (b!=0))     // nếu a khác 0 hoặc b khác 0
    if ( !a )                 // phủ định a khác 0
    if (a==b)                 // nếu a đúng bằng b
```

Các toán tử thao tác bít (không sử dụng cho float và double)

& : Phép hội các bít.

| : Phép tuyển các bít.

^ : Phép tuyển các bít có loại trừ.

<< : Phép dịch trái (dịch sang trái n bít giá trị 0)

>> : Phép dịch phải (dịch sang phải n bít có giá trị 0)

~ : Phép lấy phần bù.

Ví dụ 1.2: Viết chương trình kiểm tra các toán tử thao tác bít.

```
#include    <stdio.h>
#include    <conio.h>
void main(void){
        unsigned int a=3, b=5, c; clrscr();
        c = a & b; printf("\n c = a & b=%d",c);
        c = a | b; printf("\n c = a | b=%d",c);
        c = a ^ b; printf("\n c = a ^ b=%d",c);
        c = ~a; printf("\n c = ~a =%d",c);
        c = a << b; printf("\n c = a << b=%d",c);
        c = a >>b;  printf("\n c = a >> b=%d",c);
        getch();
}
```

Toán tử chuyển đổi kiểu: Ta có thể dùng toán tử chuyển đổi kiểu để nhận được kết quả tính toán như mong muốn. Qui tắc chuyển đổi kiểu được thực hiện theo qui tắc: (kiểu) biến.

Ví dụ 1.3: Tính giá trị phép chia hai số nguyên a và b.

```
#include        <stdio.h>
void main(void)
        int a=3, b=5; float c;
        c= (float) a / (float) b;
        printf("\n thương c = a / b =%6.2f", c);
        getch();
}
```

Thứ tự ưu tiên các phép toán : Khi viết một biểu thức, chúng ta cần lưu ý tới thứ tự ưu tiên tính toán các phép toán, các bảng tổng hợp sau đây phản ánh trật tự ưu tiên tính toán của các phép toán số học và phép toán so sánh.

Bảng tổng hợp thứ tự ưu tiên tính toán các phép toán số học và so sánh

Tên toán tử	Chiều tính toán
( ), [] , ->	***L -> R***
- , ++, -- , ! , ~ , sizeof()	***R -> L***
**** , /, %***	***L -> R***
+ , -	***L -> R***
>>, <<	***L -> R***
<, <=, > , >=,	***L -> R***
== !=	***L -> R***
&	***L -> R***
^	***L -> R***
|	***L -> R***
&&	***L -> R***
||	***L -> R***
?:	***R -> L***
=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=	***R -> L***

1.3.2. Tập các lệnh vào ra cơ bản

Nhập dữ liệu từ bàn phím: scanf("format_string, . . .", &parameter . . .);

Nhập dữ liệu từ tệp: fscanf(file_pointer,"format_string, . . .", &parameter, . . .);

Nhận một ký tự từ bàn phím: getch(); getchar();

Nhận một ký tự từ file: fgetc(file_pointer, character_name);

Nhập một string từ bàn phím: gets(string_name);

Nhận một string từ file text : fgets(string_name, number_character, file_pointer);

Xuất dữ liệu ra màn hình: printf("format_string . . .", parameter . . .);

Xuất dữ liệu ra file : fprintf(file_pointer, "format_string . . .", parameter. . .);

Xuất một ký tự ra màn hình: putch(character_name);

Xuất một ký tự ra file: fputc(file_pointer, character_name);

Xuất một string ra màn hình: puts(const_string_name);

Xuất một string ra file: fputs(file_pointer, const_string_name);

1.3.3. Thao tác trên các kiểu dữ liệu có cấu trúc

Tập thao tác trên string:

+ Cách tổ chức string và các thao tác trên string:

char *strchr(const char *s, int c) : tìm ký tự c đầu tiên xuất hiện trong xâu s;

char *stpcpy(char *dest, const char *src) : copy xâu scr vào dest;

int strcmp(const char *s1, const char *s2) : so sánh hai xâu s1 và s2 theo thứ tự từ điển, nếu s1 < s2 thì hàm trả lại giá trị nhỏ hơn 0. Nếu s1>s2 hàm trả lại giá trị dương. Nếu s1==s2 hàm trả lại giá trị 0.

char *strcat(char *dest, const char *src) : thêm xâu scr vào sau xâu dest.

char *strlwr(char *s) : chuyển xâu s từ ký tự in hoa thành ký tự in thường.

char *strupr(char *s): chuyển xâu s từ ký tự thường hoa thành ký tự in hoa.

char *strrev(char *s): đảo ngược xâu s.

char *strstr(const char *s1, const char *s2): tìm vị trí đầu tiên của xâu s2 trong xâu s1.

int strlen(char *s): cho độ dài của xâu ký tự s.

Tập thao tác trên con trỏ:

Thao tác lấy địa chỉ của biến: & parameter_name;

Thao tác lấy nội dung biến (biến có kiểu cơ bản): *pointer_name;

Thao tác trỏ tới phần tử tiếp theo: ++pointer_name;

Thao tác trỏ tới phần tử thứ n kể từ vị trí hiện tại:

pointer_name = pointer_name +n;

Thao tác trỏ tới phần tử sau con trỏ kể từ vị trí hiện tại: --pointer_name;

Thao tác trỏ tới phần tử sau n phần tử kể từ vị trí hiện tại:

Pointer_name = pointer_name - n;

Thao tác cấp phát bộ nhớ cho con trỏ: void *malloc(size_t size);void *calloc(size_t nitems, size_t size);

Thao tác cấp phát lại bộ nhớ cho con trỏ : void *realloc(void *block, size_t size);

Thao tác giải phóng bộ nhớ cho con trỏ: void free(void *block);

Tập thao tác trên cấu trúc:

Định nghĩa cấu trúc:

```
struct struct_name{
    type_1 parameter_name_1;
    type_2 parameter_name_2;
    . . . . . .
    type_k parameter_name_k;
} struct_parameter_name;
```

Phép truy nhập tới thành phần cấu trúc: struct_parameter_name.parameter_name.

Phép gán hai cấu trúc cùng kiểu:

struct_parameter_name1 = struct_parameter_name2;

Phép tham trỏ tới thành phần của con trỏ cấu trúc:

pointer_struct_parameter_name -> struct_parameter_name.

Tập thao tác trên file:

Khai báo con trỏ file: FILE * file_pointer;

Thao tác mở file theo mode: FILE *fopen(const char *filename,const char *mode);

Thao tác đóng file : int fclose(FILE *stream);

Thao tác đọc từng dòng trong file: char *fgets(char *s, int n, FILE *stream);

Thao tác đọc từng khối trong file:

size_t fread(void *ptr, size_t size,size_t n, FILE *stream);

Thao tác ghi từng dòng vào file: int fputs(const char *s, FILE *stream);

Thao tác ghi từng khối vào file:

size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);

Thao tác kiểm tra sự tồn tại của file: int access(const char *filename, int amode);

Thao tác đổi tên file: int rename(const char *oldname,const char *newname);

Thao tác loại bỏ file: int unlink(const char *filename);

1.4. Nguyên lý địa phương

- *Các biến địa phương trong hàm, thủ tục hoặc chu trình cho dù có trùng tên với biến toàn cục thì khi xử lý biến đó trong hàm hoặc thủ tục vẫn không làm thay đổi giá trị của biến toàn cục.*
- *Tên của các biến trong đối của hàm hoặc thủ tục đều là hình thức.*
- *Mọi biến hình thức truyền theo trị cho hàm hoặc thủ tục đều là các biến địa phương.*
- *Các biến khai báo bên trong các chương trình con, hàm hoặc thủ tục đều là biến địa phương.*
- *Khi phải sử dụng biến phụ nên dùng biến địa phương và hạn chế tối đa việc sử dụng biến toàn cục để tránh xảy ra các hiệu ứng phụ.*

Ví dụ hoán đổi giá trị của hai số a và b sau đây sẽ minh họa rõ hơn về nguyên lý địa phương.

Ví dụ 1.4. Hoán đổi giá trị của hai biến a và b.

```
#include        <stdio.h>
int  a, b; // khai báo a, b là hai biến toàn cục.
void  Swap(void) {
            int a,b, temp; // khai báo  a, b là hai biến địa phương
            a= 3; b=5; // gán giá trị cho a và b
            temp=a; a=b; b=temp;// đổi giá trị của a và b
            printf("\n Kết quả thực hiện trong thủ tục a=%5d b=%5d:,a,b);
}
void main(void) {
    a=1; b=8; // khởi đầu giá trị cho biến toàn cục a, b.
    Swap();
    printf("\n Kết quả sau khi thực hiện thủ tục a =%5d b=%5d",a,b);
    getch();
}
```

Kết quả thực hiện chương trình:

Kết quả thực hiện trong thủ tục a = 5 b=3

Kết quả sau khi thực hiện thủ tục a = 1 b =8

Trong ví dụ trên a, b là hai biến toàn cục, hai biến a, b trong thủ tục Swap là hai biến cục bộ. Các thao tác trong thủ tục Swap gán cho a giá trị 3 và b giá trị 5 sau đó thực hiện đổi giá trị của a =5 và b =3 là công việc xử lý nội bộ của thủ tục mà không làm thay đổi giá trị của biến toàn cục của a, b sau thi thực hiện xong thủ tục Swap. Do vậy, kết quả sau khi thực hiện Swap a = 1, b =8; Điều đó chứng tỏ trong thủ tục Swap chưa bao giờ sử dụng tới hai biến toàn cục a và b. Tuy nhiên, trong ví dụ sau, thủ tục Swap lại làm thay đổi giá trị của biến toàn cục a và b vì nó thao tác trực tiếp trên biến toàn cục.

Ví dụ 1.5. Đổi giá trị của hai biến a và b

```
#include      <stdio.h>
#include      <conio.h>
#include      <stdlib.h>
#include      <io.h>
int  a, b; // khai báo a, b là hai biến toàn cục.
void  Swap(void) {
            int temp; // khai báo  a, b là hai biến địa phương
            a= 3; b=5; // gán giá trị cho a và b
            temp=a; a=b; b=temp;// đổi giá trị của a và b
            printf("\n Kết quả thực hiện trong thủ tục a=%5d b=%5d:,a,b);
}
void main(void) {
    a=1; b=8; // khởi đầu giá trị cho biến toàn cục a, b.
    Swap();
    printf("\n Kết quả sau khi thực hiện thủ tục a =%5d b=%5d",a,b);
    getch();
}
```

Kết quả thực hiện chương trình:

Kết quả thực hiện trong thủ tục a = 8 b=1

Kết quả sau khi thực hiện thủ tục a = 1 b =8

1.5. Nguyên lý nhất quán

- *Dữ liệu thế nào thì phải thao tác thế ấy. Cần sớm phát hiện những mâu thuẫn giữa cấu trúc dữ liệu và thao tác để kịp thời khắc phục.*

Như chúng ta đã biết, kiểu là một tên chỉ tập các đối tượng thuộc miền xác định cùng với những thao tác trên nó. Một biến khi định nghĩa bao giờ cũng thuộc một kiểu xác định nào đó hoặc là kiểu cơ bản hoặc kiểu do người dùng định nghĩa. Thao tác với biến phụ thuộc vào những thao tác được phép của kiểu. Hai kiểu khác nhau được phân biệt bởi tên, miền xác định và các phép toán trên kiểu dữ liệu. Tuy nhiên, trên thực tế có nhiều lỗi nhập nhằng giữa phép toán và cấu trúc dữ liệu mà chúng ta cần hiểu rõ.

Đối với kiểu ký tự, về nguyên tắc chúng ta không được phép thực hiện các phép toán số học trên nó, nhưng ngôn ngữ C luôn đồng nhất giữa ký tự với số nguyên có độ lớn 1 byte. Do vậy, những phép toán số học trên các ký tự thực chất là những phép toán số học trên các số nguyên. Chẳng hạn, những thao tác như trong khai báo dưới đây là được phép:

```
char x1='A', x2 ='z';
x1 = (x1 + 100) % 255;
x2 = (x2-x1) %255;
```

Mặc dù x1, x2 được khai báo là hai biến kiểu char, nhưng trong thao tác

x1 = (x1 + 100) % 255;

x2 = (x2 +x1) %255;

chương trình dịch sẽ tự động chuyển đổi x1 thành mã của ký tự 'A' là 65, x2 thành mã ký tự 'z' là 122 để thực hiện phép toán. Kết quả nhận được x1 là một ký tự có mã là (65+100)%255 = 165; x2 là ký tự có mã là 32 ứng với mã của ký tự space.

Chúng ta có thể thực hiện được các phép toán số học trên kiểu int, long, float, double. Nhưng đối với int và long, chúng ta cần đặc biệt chú ý phép chia hai số nguyên cho ta một số nguyên, tích hai số nguyên cho ta một số nguyên, tổng hai số nguyên cho ta một số nguyên mặc dù thương hai số nguyên là một số thực, tích hai số nguyên hoặc tổng hai số nguyên có thể là một số long int. Do vậy, muốn nhận được kết quả đúng, chúng ta cần phải chuyển đổi các biến thuộc cùng một kiểu trước khi thực hiện phép toán. Ngược lại, ta không thể lấy modul của hai số thực hoặc thực hiện các thao tác dịch chuyển bít trên nó, vì những thao tác đó không nằm trong định nghĩa của kiểu.

Điều tương tự cũng xảy ra với các string. Trong Pascal, phép toán so sánh hai string hoặc gán trực tiếp hai Record cùng kiểu với nhau là được phép, ví dụ : Str1>Str2, Str1 := Str2; Nhưng trong C thì các phép toán trên lại không được định nghĩa, nếu muốn thực hiện nó, chúng ta chỉ có cách định nghĩa lại hoặc thực hiện nó thông qua các lời gọi hàm. Ví dụ đơn giản sau sẽ minh họa cho những lỗi thường xảy ra sự nhập nhằng giữa phép toán và cấu trúc dữ liệu.

Ví dụ 1.6. Viết chương trình tính tổng, hiệu, tích, thương của hai số nguyên a và b.

```
#include      <stdio.h>
long int tong( int a, int b) { return(a+b); }
int hieu(int a, int b) { return(a-b); }
long int tich(int a, int b) {   return(a*b);}
float  thuong(int a, int b){   return(a/b);}
void main(void){
        int a=30000, b = 20000;// khai báo và gán giá trị hai số nguyên a, b
        printf("\n Tổng hai số nguyên a + b =%ld", tong(a,b));
        printf("\n Hiệu hai số nguyên a - b =%d",hieu(a,b));
        printf("\n Tích hai số nguyên a*b=%ld", tich(a,b));
        printf('\n Thương hai số nguyên a/b =%f", thuong(a,b));
        getch();
}
```

Trong ví dụ trên, hàm tong(30000,20000) cho ta một số unsigned int là giá trị là tổng của hai số nguyên dương, vì tổng của hai số nguyên dương có thể vượt quá kích cỡ kiểu int để trở thành một số unsigned int, điều đó cũng xảy ra tương tự đối với hàm tich(30000, 20000); Do đó, việc thực hiện việc gán một số nguyên dương thế này sẽ cho ta kết quả không mong muốn.

Với hàm thương(a,b) thì hoàn toàn ngược lại đối với một số ngôn ngữ (C, C++). Khi chúng ta phát biểu thương của hai số nguyên dương là một số thực được tính là a/b (b<>0), nhưng trong thực tế chương trình dịch của C lại hiểu thương của hai số nguyên cho ta kết quả là một số nguyên, do đó chúng ta nhận được kết quả không mong muốn. Giải pháp để khắc phục mâu thuẫn giữa cấu trúc dữ liệu và thao tác trên cấu trúc dữ liệu có thể thực hiện bằng cách chuyển đổi kiểu của đối truyền cho hàm như phiên bản sau.

Ví dụ 1.7. Viết chương trình tính tổng, hiệu, tích, thương của hai số nguyên a và b.

```
#include      <stdio.h>
long int tong( int a, int b) { return((long) a+ (long) b); }
int hieu(int a, int b) { return(a-b); }
long int tich(int a, int b) {   return((long) a* (long) b);}
float  thuong(int a, int b){
        float k; k = (float) a / (float) b;
        return(k);
}
void main(void){
        int a=30000, b = 20000;// khai báo và gán giá trị hai số nguyên a, b
        printf("\n Tổng hai số nguyên a + b =%ld", tong(a,b));
        printf("\n Hiệu hai số nguyên a - b =%d",hieu(a,b));
        printf("\n Tích hai số nguyên a*b=%ld", tich(a,b));
        printf('\n Thương hai số nguyên a/b =%f", thuong(a,b));
        getch();
}
```

Kết quả thực hiện chương trình:

```
Tổng hai số nguyên a+b      = 50000
Hiệu hai số nguyên a-b      = 10000
Tích hai số nguyên a*b      = 6000000000
Hiệu hai số nguyên a/b      = 1.500000
```

Tóm lại, cần nắm vững nguyên tắc, định nghĩa và những qui định riêng của ngôn ngữ cho từng kiểu dữ liệu và các phép toán trên nó để đảm bảo tính nhất quán trong khi xử lý dữ liệu.

1.6. Nguyên lý an toàn

- *Lỗi nặng nhất nằm ở mức cao nhất (mức ý đồ thiết kế) và ở mức thấp nhất thủ tục phải chịu tải lớn nhất.*
- *Mọi lỗi, dù là nhỏ nhất cũng phải được phát hiện ở một bước nào đó của chương trình. Quá trình kiểm tra và phát hiện lỗi phải được thực hiện trước khi lỗi đó hoành hành.*

Các loại lỗi thường xảy ra trong khi viết chương trình có thể được tổng kết lại như sau:

Lỗi được thông báo bởi từ khoá error (lỗi cú pháp): loại lỗi này thường xảy ra trong khi soạn thảo chương trình, chúng ta có thể viết sai các từ khoá ví dụ thay vì viết là int chúng ta soạn thảo sai thành Int (lỗi chữ in thường thành in hoa), hoặc viết sai cú pháp các biểu thức như thiếu các dấu ngoặc đơn, ngoặc kép hoặc dấu chấm phảy khi kết thúc một lệnh, hoặc chưa khai báo nguyên mẫu cho hàm .

Lỗi được thông báo bởi từ khoá Warning (lỗi cảnh báo): lỗi này thường xảy ra khi ta khai báo biến trong chương trình nhưng lại không sử dụng tới chúng, hoặc lỗi trong các biểu thức kiểm tra khi biến được kiểm tra không xác định được giá trị của nó, hoặc lỗi do thứ tự ưu tiên các phép toán trong biểu thức. Hai loại lỗi error và warning được thông báo ngay khi dịch chương trình thành file *.OBJ. Quá trình liên kết (linker) các file *.OBJ để tạo nên file chương trình mã máy *.EXE chỉ được tiếp tục khi chúng ta hiệu đính và khử bỏ mọi lỗi error.

Lỗi xảy ra trong quá trình liên kết: lỗi này thường xuất hiện khi ta sử dụng tới các lời gọi hàm , nhưng những hàm đó mới chỉ tồn tại dưới dạng nguyên mẫu (function prototype) mà chưa được mô tả chi tiết các hàm, hoặc những lời hàm gọi chưa đúng với tên của nó. Lỗi này được khắc phục khi ta bổ sung đoạn chương trình con mô tả chi tiết cho hàm hoặc sửa đổi lại những lời gọi hàm tương ứng.

Ta quan niệm, lỗi cú pháp (error), lỗi cảnh báo (warning) và lỗi liên kết (linker) là lỗi tầm thường vì những lỗi này đã được Compiler của các ngôn ngữ lập trình phát hiện được. Để khắc phục các lỗi loại này, chúng ta chỉ cần phải đọc và hiểu được những thông báo lỗi thường được viết bằng tiếng Anh. Cũng cần phải lưu ý rằng, do mức độ phức tạp của chương trình dịch nên không phải lỗi nào cũng được chỉ ra một cách tường minh và chính xác hoàn toàn tại nơi xuất hiện lỗi.

Loại lỗi cuối cùng mà các compiler không thể phát hiện nổi đó là lỗi do chính lập trình viên gây nên trong khi thiết kế chương trình và xử lý dữ liệu. Những lỗi này không được compiler thông báo mà nó phải trả giá bằng quá trình tự test hoặc chứng minh được tính đúng đắn của chương trình. Lỗi có thể nằm ở chính ý đồ thiết kế, hoặc lỗi do không lường trước được tính chất của mỗi loại thông tin vào.

1.6. Phương pháp Top-Down

- *Quá trình phân tích bài toán được thực hiện từ trên xuống dưới. Từ vấn đề chung nhất đến vấn đề cụ thể nhất. Từ mức trừu tượng mang tính chất tổng quan tới mức đơn giản nhất là đơn vị chương trình.*

Một trong những nguyên lý quan trọng của lập trình cấu trúc là phương pháp phân tích từ trên xuống (Top - Down) với quan điểm "thấy cây không bằng thấy rừng", phải đứng cao hơn để quan sát tổng thể khu rừng chứ không thể đứng trong rừng quan sát chính nó.

Quá trình phân rã bài toán được thực hiện theo từng mức khác nhau. Mức thấp nhất được gọi là mức tổng quan (level 0), mức tổng quan cho phép ta nhìn tổng thể hệ