

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG
TIN HỌC CƠ SỞ 2

KHOA PHỤ TRÁCH: Khoa CNTT1
CHỦ BIÊN: TS. PHAN THỊ HÀ

Hà Nội – Năm 2020

1

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

MỤC LỤC

Contents

1. GIỚI THIỆU CHUNG	4
1.1. Ngôn ngữ lập trình	4
1.2. Thuật toán (Algorithm)	6
1.3. Sự ra đời và phát triển của ngôn ngữ	

C.....	7
2. MỘT SỐ KIẾN THỨC CƠ SỞ.....	7
2.1. Bộ kí tự, từ khóa,tên.....	7
2.1.1 Bộ kí tự trong C.....	7
2.1.2 Các từ khoá (Keywords).....	7
2.1.3 Tên và cách đặt tên.....	8
2.1.4 Lời giải thích.....	8
2.2. Cấu trúc chương trình trong C	8
2.2.1 Cấu trúc tổng quát của chương trình trong C.....	8
2.2.2 Chương trình đơn giản nhất trong C	9
2.3. Các kiểu dữ liệu cơ sở.....	10
2.4. Biến,hằng, câu lệnh và các phép toán	11
2.4.1 Biến và hằng.....	11
2.4.2 Câu lệnh.....	12
2.4.3 Các phép toán	13
2.5. Thủ tục vào và ra chuẩn	16
2.5.1 Vào ra bằng getchar(), putchar()	16
2.5.2 In ra theo khuôn dạng - Printf.....	17
2.5.3 Nhập vào có khuôn dạng - scanf	19
2.5.4 Thâm nhập vào thư viện chuẩn	21
3. CÁC CẤU TRÚC LỆNH ĐIỀU KHIỂN.....	25
3.1. Câu lệnh khối.....	25
3.2. Cấu trúc lệnh if.....	25
3.3. Cấu trúc lệnh switch.....	27
3.4. Vòng lặp for	28
3.5. Vòng lặp không xác định while Cú pháp:.....	30
3.6. Vòng lặp không xác định do . . while Cú pháp:.....	31
3.7. Lệnh break và lệnh Continue	33
4. HÀM VÀ PHẠM VI HOẠT ĐỘNG CỦA BIẾN.....	44
4.1. Tính chất của hàm	44
4.2. Khai báo, thiết kế hàm	44
4.3. Phương pháp truyền tham biến cho hàm.....	47
4.4. Biến địa phương, biến toàn cục.....	49
4.5. Tính đệ qui của hàm.....	52
5. CẤU TRÚC DỮ LIỆU KIỂU MẢNG (Array).....	55
5.1. Khái niệm về mảng	55
5.2. Các thao tác đối với mảng.....	58
5.3. Mảng và đối của hàm.....	61

5.4. Xâu kí tự (string).....	64
5.5. Kiểu dữ liệu Con trỏ.....	70
5.5.1 Con trỏ và địa chỉ	70
5.5.2 Con trỏ và đối của hàm.....	72
5.5.3 Con trỏ và mảng	72
5.5.4 Cấp phát bộ	

trở.....	76
trở.....	90
main().....	92
hàm.....	93
tệp (FILE).....	105
vào thư viện chuẩn	105
tệp.....	108
Stderr và Exit.....	111
đưa ra cả dòng.....	112
file bằng fscanf, fprintf.....	113
thông dụng khác	117
(STRUCT).....	120
cấu trúc	120
dụng cấu trúc	121
typedef khi làm việc với cấu trúc.....	121
thành phần của cấu trúc:	122
hàm.....	123
.....	124
trở	125
5.6. Mảng các con	
5.7. Đối của hàm	
5.8. Con trở	
6. Dữ liệu kiểu	
6.1. Thâm nhập	
6.2. Thâm nhập	
6.3. Xử lý lỗi -	
6.4. Đưa vào và	
6.5. Đọc và ghi	
6.6. Một số hàm	
7. CẤU TRÚC	
7.1. Định nghĩa	
7.2. Khai báo và sử	
7.2.1. Sử dụng từ khóa	
7.2.2. Truy cập vào các	
7.3. Cấu trúc và	
7.4. Cấu trúc lồng	
7.5. Cấu trúc và con	
7.6. Cấu trúc và file	

3

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

NGÔN NGỮ LẬP TRÌNH C

MỞ ĐẦU

Tin học cơ sở 2 là môn học quan trọng trong chương trình giáo dục đại cương ở

bậc

đại

học,

đây

là

môn

bắt

buộc

đối

với

tất cả các sinh viên trong Học Viện CNBCVT. Tài liệu này nhằm cung cấp cho sinh viên các kiến thức tổng quan và cơ bản về ngôn ngữ lập trình C. Qua đó học viên có thể nắm được các khái niệm cơ bản về lập trình và thiết lập được một số chương trình đơn giản phục vụ cho khoa học kỹ thuật và đặc biệt là làm công cụ để phục vụ cho các môn học về tin học và viễn thông mà các em sắp học. Chúng tôi đã biên soạn bài giảng này cho tất cả các sinh viên các ngành kỹ thuật ở bậc đại học với mục đích giúp cho các sinh viên có một tài liệu học cần thiết cho môn học này và cũng để đáp ứng nhu cầu càng ngày càng cao về tư liệu dạy và học tin học.

1. GIỚI THIỆU CHUNG

1.1. Ngôn ngữ lập trình

Trong phần “Tin học cơ sở 1” chúng ta đã tìm hiểu Winword và Excel, là các phần mềm ứng dụng trong công việc soạn thảo văn bản và làm các bảng tính toán được. Đặc điểm của các phần mềm ứng dụng là luôn định rõ phạm vi ứng dụng và cung cấp càng nhiều càng tốt các công cụ để hoàn thành chức năng đó. Tuy nhiên người sử dụng cũng hầu như bị bó buộc trong phạm vi quy định của phần mềm. Chẳng hạn ta khó có thể dùng Excel để giải một bài toán gồm nhiều bước tính toán như tính nghiệm gần đúng một phương trình vi phân hay giải một hệ phương trình tuyến tính. Mặc dầu các phần mềm ứng dụng ngày càng nhiều và thuộc đủ các lĩnh vực như xây dựng, thiết kế, hội họa, âm nhạc...nhưng không thể bao trùm hết các vấn đề nảy sinh trong thực tế vô cùng phong phú. Rõ ràng không chỉ những chuyên gia tin học mà ngay cả những người sử dụng, nhất là các cán bộ kỹ thuật, rất cần đến những phần mềm uyên chuyên và mềm dẻo hơn, có khả năng thực hiện được nhiều hơn các chỉ thị của người sử dụng để giúp họ giải quyết những công việc đa dạng bằng máy tính. Phần mềm có tính chất như thế được gọi là ngôn ngữ lập trình. Chính xác hơn ngôn ngữ lập trình là một ngôn ngữ nhân tạo bao gồm một tập các từ vựng (mà ta sẽ gọi là từ khóa để phân biệt với ngôn ngữ thông thường) và một tập các quy tắc (gọi là Syntax - cú pháp) mà ta có thể sử dụng để biên soạn các lệnh cho máy tính thực hiện.

Như ta đã biết, các ô nhớ của máy tính chỉ có thể biểu diễn các số 0 và 1. Vì vậy ngôn

ngữ mà máy có thể hiểu trực tiếp là ngôn ngữ trong đó các lệnh là các dãy số nhị phân và do đó được gọi là ngôn ngữ máy (machine language). Mọi ngôn ngữ khác đều phải thông dịch hoặc biên dịch sang ngôn ngữ máy (Interpreter - thông dịch và cho chạy từng lệnh. Compiler - biên dịch thành 1 chương trình ngôn ngữ máy hoàn chỉnh, do vậy chạy nhanh hơn thông dịch).

Có nhiều loại ngôn ngữ lập trình, và hầu hết các nhà khoa học về máy tính đều cho rằng không có một ngôn ngữ độc nhất nào có đủ khả năng phục vụ cho các yêu cầu của tất cả các

viên. Theo truyền thống, các ngôn ngữ lập trình được phân làm 2 loại: các ngôn ngữ bậc thấp và ngôn ngữ bậc cao.

Ngôn ngữ lập trình bậc thấp (low-level programming language):

Ngôn ngữ máy, hợp ngữ (assembler: chương trình dịch hợp ngữ, assembly language: ngôn ngữ hợp ngữ). Hợp ngữ là ngôn ngữ lập trình bậc thấp từ ngôn ngữ máy. Nó chỉ khác với ngôn ngữ máy trong việc sử dụng các mã biểu thị các chức năng chính mà máy thực hiện.

Lập trình bằng hợp ngữ rất phiền toái: có đến vài tá dòng mã cần thiết chỉ để thực hiện phép cộng 2 con số. Các chương trình hợp ngữ rất khó viết; chúng không có cấu trúc hoặc modun hóa rõ ràng. Chương trình hợp ngữ cũng không dễ chuyển từ loại máy tính này sang loại máy tính khác. Các chương trình này được viết theo các tập lệnh đặc thù của loại bộ vi xử lý nhất định. Lập trình bằng hợp ngữ thì mã gọn và chạy nhanh. Do đó hầu hết các chương trình điều hành hệ thống đều được viết bằng hợp ngữ. Tuy nhiên do sự phức tạp của công việc lập trình nên các hãng sản xuất phần mềm chuyên dụng thích viết chương trình bằng ngôn ngữ C (do Bell Laboratories của hãng AT&T xây dựng) là loại ngôn ngữ kết hợp được cấu trúc của ngôn ngữ bậc cao hiện đại với tốc độ và tính hiệu quả của hợp ngữ bằng cách cho phép nhúng các lệnh hợp ngữ vào chương trình.

Ngôn ngữ lập trình bậc cao:

Các ngôn ngữ lập trình bậc cao như Basic, Pascal, C, C++... cho phép các lập trình viên có thể diễn đạt chương trình bằng các từ khóa và các câu lệnh gần giống với ngôn ngữ tự nhiên. Các ngôn ngữ này được gọi là “bậc cao” vì chúng giải phóng các lập trình viên khỏi những quan tâm về từng lệnh sẽ được máy tính tiến hành như thế nào, bộ phận thông dịch hoặc biên dịch của chương trình sẽ giải quyết các chi tiết này khi mã nguồn được biến đổi thành ngôn ngữ máy. Một câu lệnh trong ngôn ngữ bậc cao tương ứng với một số lệnh ngôn ngữ máy, cho nên bạn có thể thảo luận theo ngôn ngữ bậc cao nhanh hơn so với bậc thấp. Tuy nhiên bạn cũng phải trả giá cho việc này. Chương trình ngôn ngữ máy được dịch ra từ mã nguồn được viết bằng ngôn ngữ bậc cao chứa rất nhiều chi tiết thừa, do đó tốc độ chạy sẽ chậm hơn nhiều so với chương trình viết bằng hợp ngữ. Thông thường một trình biên dịch đặc trưng thường sinh ra số lệnh mã máy gấp 2 lần hay nhiều hơn số lệnh cần thiết nếu viết bằng mã máy.

Một cách phân loại khác của các ngôn ngữ lập trình:

Ngôn ngữ thủ tục (Procedural Language) và ngôn ngữ khai báo (Declarative Language)

Ngôn ngữ thủ tục: Lập trình viên phải xác định một thủ tục mà máy tính sẽ tuân theo để hoàn thành một công việc định trước. Thí dụ: Basic, C, Fortran, ...

Ngôn ngữ khai báo: Ngôn ngữ sẽ định nghĩa một loạt các yếu tố và các quan hệ, đồng thời cho phép bạn có thể tiến hành xếp hàng đối với những kết quả xác định. Thí dụ: Prolog, SQL (Structured Query Language)

Điều then chốt trong việc lập trình chuyên dụng là môdun hóa ngôn ngữ - đó là sự phát

nhiệm vụ lập trình có thể phân phối được cho các thành viên của một nhóm lập trình, và kết quả đạt được là các bộ phận khác nhau sẽ hoạt động phù hợp với nhau khi nhiệm vụ của từng người hoàn thành. Ngôn ngữ lập trình môđun, như Module-2 hoặc ngôn ngữ hướng đối tượng như C++, sẽ cho phép từng lập trình viên có thể tập trung vào việc lập mã, biên dịch và gỡ rối các module chương trình riêng biệt, đồng thời có thể cho chạy (kiểm tra thử) riêng từng module của mình. Khi từng module riêng đã chạy tốt chúng sẽ được liên kết với nhau mà không gây trục trặc nào.

1.2. Thuật toán (Algorithm)

Thuật ngữ Algorithm được dịch ra tiếng Việt là thuật toán, thuật giải hoặc giải thuật. Ở đây dùng từ thuật toán là cách gọi quen thuộc với nhiều người.

Thuật toán là một dãy hữu hạn các bước, mỗi bước mô tả chính xác các phép toán hoặc hành động cần thực hiện, để giải quyết một vấn đề.

Để hiểu đầy đủ ý nghĩa của khái niệm thuật toán, chúng ta nêu ra 6 đặc trưng sau đây của thuật toán:

Input Mỗi thuật toán thường có một số dữ liệu vào.

Output Mỗi thuật toán thường có một số dữ liệu ra.

Tính xác định (Definiteness) Mỗi bước được mô tả chính xác, chỉ có một cách hiểu duy nhất và đủ đơn giản để có thể thực hiện được.

Tính dừng (Finiteness) Thuật toán phải dừng sau một số hữu hạn bước thực hiện

Tính hiệu quả (Effectiveness) Các phép toán trong các bước phải đủ đơn giản để có thể thực hiện được.

Tính tổng quát (Generalness) Thuật toán phải có tính tổng quát, có thể áp dụng cho một lớp đối tượng.

Ví dụ:

Thuật toán Euclid: Tìm ước số chung lớn nhất của hai số tự nhiên m, n .

Input: m, n nguyên dương

Output: g là ước số chung lớn nhất của m và n

Phương pháp:

1. $r := m \bmod n$

6

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

2. Nếu $r=0$ thì $g:=n$

Ngược lại ($r>0$) $m:=n$; $n:=r$ và quay lại bước 1.

1.3. Sự ra đời và phát triển của ngôn ngữ C

Năm 1970 Ken Thompson sáng tạo ra ngôn ngữ B dùng trong môi trường hệ điều hành UNIX trên máy điện toán DEC PD-7. B cũng là chữ tắt của BCPL (Basic Combined Programming Language) do Martin Richards viết. Năm 1972 Dennis Ritchie của hãng Bell

Laboratories (và Ken Thompson) sáng tạo nên ngôn ngữ C nhằm tăng hiệu quả cho ngôn ngữ B. Lúc đầu ngôn ngữ C không được mọi người ưa dùng. Nhưng sau khi D.Ritchie cho xuất bản cuốn "The C Programming Language" ("Ngôn ngữ lập trình C") thì ngôn ngữ C được chú ý và được sử dụng rộng rãi. Người ta đã dùng C để viết hệ điều hành đa nhiệm UNIX, O/S 2 và ngôn ngữ Dbase. C đã được cải tiến qua nhiều phiên bản: trình biên dịch Turbo C từ phiên bản 1 đến phiên bản 5, Microsoft C từ phiên bản 1 đến phiên bản 6. Hiện nay, C lại được phát triển để thành C++ với 3 trình biên dịch: Borland C++, Visual C++ và Turbo C++.

Mặc dù hiện nay có khá nhiều ngôn ngữ lập trình mới, nhưng C vẫn là một ngôn ngữ lập trình được ưa chuộng. C được ứng dụng để viết các phần mềm trong nhiều lĩnh vực, đặc biệt là trong khoa học kỹ thuật.

2. MỘT SỐ KIẾN THỨC CƠ SỞ

2.1. Bộ kí tự, từ khoá, tên

2.1.1 Bộ kí tự trong C

Mọi ngôn ngữ đều được xây dựng trên một bộ kí tự (các chữ, các kí hiệu). Đối với ngôn ngữ C sử dụng bộ kí tự sau:

Tập các chữ cái in hoa: A, B, C, D, . . , Z

Tập các chữ cái in thường: a, b, c, d, . . , z

Tập các chữ số: 0, 1, 2, 3, . . , 9

Các dấu chấm câu: , . ; : / ? [] { } ! @ # \$ ^ & * () + = - < > "

Các kí tự không nhìn thấy: dấu trống (Space), dấu Tab, dấu xuống dòng (Enter),

Dấu gạch dưới _

2.1.2 Các từ khoá (Keywords)

Từ khoá là tập các từ dùng riêng của ngôn ngữ, mỗi từ khoá mang theo một ý nghĩa và tác dụng riêng. Từ khoá không thể định nghĩa lại và cũng không thể lấy từ khoá đặt tên cho các đối tượng. Dưới đây là bảng liệt kê các từ khoá thông dụng trong C.

auto break base char continue default do double else extern float for

7

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

goto if int long register return short sizeof static struct switch typedef union

unsigned void public while volatile

2.1.3 Tên và cách đặt tên

Tên hay còn gọi là định danh (identifier) dùng để gọi các biến, hằng hoặc hàm. Đối với

ngôn
ngữ

C, mọi tên phải được khai báo trước khi sử dụng. Tên là dãy các kí tự liên nhau gồm các chữ cái, a . . z, A . . Z, các chữ số 0 . . 9 và dấu gạch dưới (dấu gạch dưới thường dùng để liên kết các thành phần của tên). Tuy nhiên, tên không được bắt đầu bằng chữ số và không chứa các kí tự đặc biệt như dấu cách, dấu tab và dấu chấm câu. Không được lấy từ khoá của C để đặt tên cho đối tượng.

Ví dụ về cách đặt tên đúng: Delta, E_Mu_X, Function1 . . .

Ví dụ về cách đặt tên sai:

2Delta: bắt đầu bằng kí tự số

E Mu_X: chứa khoảng trắng

Ngôn ngữ C phân biệt chữ in hoa và chữ in thường, do vậy những tên sau đây là khác nhau: x <> X, While <> while, For <> for. Do vậy, chúng ta cần lưu ý trong khi viết chương trình. Thông thường tên các biến, hàm được đặt bằng chữ in thường, tên các hằng được đặt bằng chữ in hoa.

2.1.4 Lời giải thích

Trong khi viết chương trình, đôi khi chúng ta cần ghi thêm một số lời ghi chú hoặc giải thích để chương trình trở nên dễ hiểu và dễ đọc. Lời giải thích không có tác dụng tạo nên mã chương trình và sẽ được trình dịch bỏ qua trong khi dịch chương trình. Phần ghi chú có thể biểu hiện trên nhiều dòng và được đánh dấu bởi cặp kí hiệu */* đoạn văn bản ghi chú */*.

2.2. Cấu trúc chương trình trong C

2.2.1 Cấu trúc tổng quát của chương trình trong C

Chương trình tổng quát viết bằng ngôn ngữ C được chia thành 6 phần, trong đó có một số phần có thể có hoặc không có tùy thuộc vào nội dung chương trình và ý đồ của mỗi lập trình viên.

Phần 1: Khai báo các chỉ thị đầu tệp và định nghĩa các hằng sử dụng trong chương trình.

Phần 2: Định nghĩa các cấu trúc dữ liệu mới (user type) để sử dụng trong khi viết chương trình.

Phần 3: Khai báo các biến ngoài (biến toàn cục) được sử dụng trong chương trình. **Phần**

4: Khai báo nguyên mẫu cho hàm (Function Ptototype). Nếu khai báo qui cách

xây dựng và chuyển tham biến cho hàm, compiler sẽ tự động kiểm tra giữa nguyên mẫu của hàm có phù hợp với phương thức xây dựng hàm hay không trong văn bản chương trình.

Phần 5: Mô tả chi tiết các hàm, các hàm được mô tả phải phù hợp với nguyên mẫu đã được khai báo cho hàm.

Phần 6: Hàm *main()*, hàm xác định điểm bắt đầu thực hiện chương trình và điểm kết

thúc
thực
hiện

chương trình.

2.2.2 Chương trình đơn giản nhất trong C

Ví dụ: Viết chương trình in ra dòng thông báo "Ngôn ngữ lập trình C".

```
#include <stdio.h>

/* khai báo việc sử dụng các hàm printf(), getch() trong conio.h*/

int main()

{
    printf ("Ngôn ngữ lập trình C\ n");/* in ra màn hình*/
    return 0;
}
```

Kết quả thực hiện chương trình: Dòng chữ được in ra

Ngôn ngữ lập trình C

Để tiếp tục hãy bấm tiếp một phím bất kì ta sẽ trở về với trình soạn thảo trong Turbo C.

Chỉ thị `#include` được gọi là chỉ thị tiền xử lý, có nhiệm vụ liên kết với tệp tương ứng

được đặt trong hai kí tự < tên file đầu tệp >. File có dạng *.h được C qui định là các file chứa nguyên mẫu của các hàm và thường được đặt trong thư mục C:\TC\INCLUDE. Như vậy, chỉ thị khai báo việc sử dụng các hàm trong file conio.h, trong trường hợp này ta sử dụng hàm printf() và getch().

Một chương trình C, với bất kì kích thước nào, cũng đều bao gồm một hoặc nhiều "hàm", trong thân của hàm có thể là các lệnh hoặc lời gọi hàm, kết thúc một lệnh là kí tự ';'. Các lời gọi hàm sẽ xác định các thao tác tính toán thực tế cần phải thực hiện. Các hàm của C cũng tương tự như các hàm và chương trình con của một chương trình FORTRAN hoặc một thủ tục PASCAL. Trong ví dụ trên main cũng là một hàm như vậy. Thông thường chúng ta được tự do chọn lấy bất kì tên nào để đặt cho hàm, nhưng main là một tên đặc biệt, chương trình sẽ được thực hiện tại điểm đầu của main. Điều này có nghĩa là mọi chương trình trong C phải có một main ở đâu đó. Main sẽ khởi động các hàm khác để thực hiện công việc của nó, một số hàm nằm ở trong văn bản chương trình, một số khác nằm ở các thư viện của các hàm đã viết trước.

Một phương pháp trao đổi dữ liệu giữa các hàm được thực hiện thông qua đối của hàm.

Các dấu ngoặc theo sau tên hàm bao quanh danh sách đối. Thông thường, mỗi hàm khi thực hiện đều trả về một giá trị, tuy nhiên cũng có hàm không có giá trị trả về. Kiểu giá trị trả về của hàm được viết đằng trước tên hàm. Nếu không có giá trị trả về thì từ khóa void

9

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

được dùng để thay thế (main là hàm không có giá trị trả về). Dấu ngoặc nhọn {} bao quanh các câu lệnh tạo nên thân của hàm, chúng tương tự như Begin . . End trong Pascal. Mọi chương trình trong C đều phải được bao trong {} và không có dấu chấm phẩy ở cuối văn bản chương trình. Hàm được khởi động thông qua tên của nó, theo sau là danh sách các đối trong ngoặc. Nếu hàm không có đối thì phải viết các dấu ngoặc tròn cho dù trong ngoặc tròn để trống.

Dòng được viết

```
printf  
("Ngôn  
ngữ lập  
trình C\  
n");
```

Là một
lời gọi
tới hàm
có tên
printf
với đối
là một
hằng
xâu kí tự
"Ngôn

ngữ lập trình C\n". printf là hàm thư viện để đưa kết quả ra trên màn hình (trừ khi xác định rõ thiết bị nhận là loại gì khác). Trong trường hợp này hàm sẽ cho hiển thị trên màn hình dãy kí tự tạo nên đối.

Dãy các kí tự bất kì nằm trong hai ngoặc kép "...." được gọi là một xâu kí tự hoặc một hằng kí tự. Hiện tại chúng ta chỉ dùng xâu kí tự như là đối của printf và một số hàm khác.

Dãy \n trong xâu kí tự trên là cú pháp của C để chỉ kí tự xuống dòng, báo hiệu lần đưa ra sau sẽ được thực hiện ở đầu dòng mới. Ngoài ra C còn cho phép dùng \t để chỉ dấu tab, \b cho việc lùi lại (backspace), \" cho dấu ngoặc kép, và \\ cho bản thân dấu sổ chéo.

2.3. Các kiểu dữ liệu cơ sở

Một kiểu dữ liệu (Data Type) được hiểu là tập hợp các giá trị mà một biến thuộc kiểu đó có thể nhận được làm giá trị của biến cùng với các phép toán trên nó. Các kiểu dữ liệu cơ sở trong C bao gồm kiểu các số nguyên (int, long), kiểu số thực (float, double), kiểu kí tự (char). Khác với Pascal, C không xây dựng nên kiểu Boolean, vì bản chất kiểu Boolean là kiểu nguyên chỉ nhận một trong hai giá trị khác 0 hoặc bằng 0.

Biến kiểu char có kích cỡ 1 byte dùng để biểu diễn 1 kí tự trong bảng mã ASCII, thực chất là số nguyên không dấu có giá trị từ 0 đến 255. Chúng ta sẽ còn thảo luận kỹ hơn về kiểu dữ liệu char trong những phần tiếp theo.

Biến kiểu số nguyên có giá trị là các số nguyên và các số nguyên có dấu (âm, dương) int, long int (có thể sử dụng từ khoá signed int, signed long), kiểu số nguyên không dấu unsigned int, unsigned long. Sự khác biệt cơ bản giữa int và long chỉ là sự khác biệt về kích cỡ.

Biến có kiểu float biểu diễn các số thực có độ chính xác đơn.

Biến có kiểu double biểu diễn các số thực có độ chính xác kép.

Sau đây là bảng các giá trị có thể của các kiểu dữ liệu cơ bản của C:

Kiểu	Miền xác định	Kích thước
char	0.. 255	1 byte
int	-32768 . . 32767	2 byte

10

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

long	-2147483648..2147483647	4 byte
unsigned int	0 . . 65535	2 byte
unsigned long	0 . . 2147483647*2=4294967295	4 byte
float	3. 4e-38 . . 3.4e + 38	4 byte
double	1.7e-308 . . 1.7e + 308	8 byte

từ sizeof(tên_kiểu) sẽ cho ta chính xác kích cỡ của kiểu tính theo byte. Chương trình sau sẽ in ra kích cỡ của từng kiểu dữ liệu cơ bản.

Ví dụ:

```
/* Chương trình kiểm tra kích cỡ các kiểu dữ liệu cơ bản */
#include <stdio.h>

int main() {
    clrscr(); /* hàm xoá toàn bộ màn hình được khai báo trong stdio.h */
    printf("\n Kích cỡ kiểu kí tự: %d", sizeof(char));
    printf("\n Kích cỡ kiểu số nguyên: %d", sizeof(int));
    printf("\n Kích cỡ kiểu số nguyên dài: %d", sizeof(long));
    printf("\n Kích cỡ kiểu số thực: %d", sizeof(float));
    printf("\n Kích cỡ kiểu số thực có độ chính xác kép: %d",
        sizeof(double));
    return 0;
}
```

2.4. Biến, hằng, câu lệnh và các phép toán

2.4.1 Biến và hằng

Biến: Biến là một đại lượng có giá trị thay đổi trong khi thực hiện chương trình. Mỗi biến có một tên và một địa chỉ của vùng nhớ dành riêng cho biến. Mọi biến đều phải khai báo trước khi sử dụng nó. Quy tắc khai báo một biến được thực hiện như sau:

Tên_kiểu_dữ_liệu tên_biến; trong trường hợp có nhiều biến có cùng kiểu, chúng ta có thể khai báo chung trên một dòng trong đó mỗi biến được phân biệt với nhau bởi một dấu phẩy và có thể gán giá trị ban đầu cho biến trong khi khai báo.

Ví dụ :

```
int a, b, c=0;
/* khai báo 3 biến a, b, c có kiểu int trong đó c được gán là 0 */
float e, f, g= 1.5; /* khai báo 3 biến e, f, g có kiểu float */
long i, j; /* khai báo i, j có kiểu long */
unsigned k,m; /* khai báo k,m có kiểu số nguyên dương */
char key; /* khai báo key có kiểu char */
```

- **Hằng :** Hằng là đại lượng mà giá trị của nó không thay đổi trong thời gian thực hiện chương trình. C sử dụng chỉ thị #define để định nghĩa các hằng.

- Hằng có giá trị trong miền xác định của kiểu int là hằng kiểu nguyên (nếu không có l ở cuối).
- Hằng có giá trị trong miền xác định của kiểu int và có kí hiệu 0x ở đầu là hằng kiểu

nguyên biểu diễn theo cơ số hệ 16 (0xFF).

▪ Hằng có giá trị trong miền xác định của kiểu long và có kí hiệu L (l) ở cuối cũng được coi là hằng kiểu long

(135L).

- Hằng có giá trị trong miền xác định của kiểu long là hằng kiểu long ▪ Hằng có giá trị trong miền xác định của kiểu float là hằng kiểu số thực (3.414).
- Hằng có giá trị là một kí tự được bao trong dấu nháy đơn được gọi là hằng kí tự ('A').
- Hằng có giá trị là một dãy các kí tự được bao trong dấu nháy kép được gọi là hằng xâu kí tự "Hằng String".

Ví dụ:

```
#define MAX 100 /* định nghĩa hằng kiểu nguyên*/
```

```
#define MIN 0xFF /* hằng nguyên biểu diễn theo cơ số hệ 16*/ #define N  
123L /* hằng long*/
```

```
#define PI 3.414 /* hằng thực*/
```

```
#define KITU 'A' /* hằng kí tự */
```

```
#define STR "XAU KI TU" /*hằng xâu kí tự*/
```


2.4.2 Câu lệnh

Câu lệnh là phần xác định công việc mà chương trình phải thực hiện để xử lý các dữ liệu đã được mô tả và khai báo. Trong C các câu lệnh cách nhau bởi dấu chấm phẩy. Câu lệnh được chia ra làm hai loại: câu lệnh đơn giản và câu lệnh có cấu trúc

Câu lệnh đơn giản là lệnh không chứa các lệnh khác như lệnh gán; lệnh gán được dùng để gán giá trị của biểu thức, một hằng vào một biến, phép gán được viết tổng quát như sau: biến= biểu thức.

Câu lệnh có cấu trúc: Bao gồm nhiều lệnh đơn giản và có khi có cả lệnh cấu trúc khác bên trong. Các lệnh loại này như :

- + Cấu trúc lệnh khối (lệnh ghép hay lệnh hợp)
- + Lệnh if
- + Lệnh switch
- + Các lệnh lặp: for, while, do.... while

12

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

2.4.3 Các phép toán

- Các phép toán số học:Gồm có: +, -, *, / (cộng, trừ, nhân, chia), % (lấy phần dư). Phép chia (/) sẽ cho lại một số nguyên giống như phép chia nguyên nếu chúng ta thực hiện chia hai đối tượng kiểu nguyên.

Ví dụ:

```
int a=3, b=5, c; /* khai báo ba biến nguyên*/
```

```
float d =3,  
e=2, f; /*  
khai báo  
ba biến  
thực*/
```

```
c = a + b;  
/* c có giá  
trị là 8*/
```

```
c = a - b;  
/* c có giá  
trị là -2*/
```

```
c = a / b ;  
/* c có giá  
trị là 0*/
```

```
c = a % b;  
/* c có giá  
trị là 3*/
```

```
f = d / e;  
/* f có giá  
trị là 1.5*/
```

Để tiện lợi trong viết chương trình cũng như giảm thiểu các kí hiệu sử dụng trong các biểu thức số học. C trang bị một số phép toán tăng và giảm mở rộng cho các số nguyên như sau:

$a++ \Leftrightarrow a = a + 1$

$a-- \Leftrightarrow a = a - 1$

$++a \Leftrightarrow a = a + 1$

$--a \Leftrightarrow a = a - 1$

$a+=n \Leftrightarrow a = a + n$

$a-=n \Leftrightarrow a = a - n$

$a/=n \Leftrightarrow a = a / n$

$a*=n \Leftrightarrow a = a * n$

$a\%=n \Leftrightarrow a = a \% n$

Chú ý: Mặc dù $++a$ và $a++$ đều tăng a lên một đơn vị, nhưng khi thực hiện các biểu thức so sánh, $++a$ sẽ tăng a trước rồi thực hiện so sánh, còn $a++$ sẽ so sánh trước sau đó mới tăng a . Tình huống sẽ xảy ra tương tự đối với $--a$ và $a--$.

Ví dụ 4.3: Kiểm tra lại các phép toán số học trên hai số nguyên a và

```
b; #include <stdio.h>

int main()
{
    int a=5, b=2;

    printf("\ tổng a + b = %d", a + b);
    printf("\ hiệu a - b = %d", a - b);
    printf("\ tích a * b = %d", a * b);
    printf("\ thương a / b = %d", a / b);
```

```
/* thương hai số nguyên sẽ là một số nguyên*/
printf("\ phần dư a % b = %d", a % b);
a++; b--; /* a = a +1; b= b-1; */
printf("\n giá trị của a, b sau khi tăng (giảm): a =%d b =%d", a, b);
a+=b; /* a=a+b;*/
```

```

printf("\n giá trị của a sau khi tăng b đơn vị: a =%d", a);
a+=b; /* a = a + b*/

printf("\n giá trị của a sau khi trừ b đơn vị: a =%d", a);
a-=b; /* a = a - b*/

printf("\n giá trị của a sau khi nhân b đơn vị: a =%d", a);
a*=b; /* a = a*b;*/

printf("\n giá trị của a sau khi chia b đơn vị: a =%d", a);
a /=b; /* a= a/b; */

printf("\n giá trị của a sau khi lấy modul b : a =%d", a);
return 0;
}

```

- Các phép toán so sánh: Gồm có các phép >, <, >=, <=, ==, != (lớn hơn, nhỏ hơn, lớn hơn hoặc bằng, nhỏ hơn hoặc bằng, đúng bằng, khác).\

Ví dụ:

```

if ( a>b) { . . } /* nếu a lớn hơn b*/

```

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

- Các phép toán logic

&& : 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.

Ví dụ:

```
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*/
```

- Các toán tử thao tác bit:

Các toán tử thao tác bit không sử dụng cho float và double:

14

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

& : Phép và (and) các bit.

| : Phép hoặc (or) các bit.

^ : Phép hoặc loại trừ bit (*XOR*).

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

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

~ : Phép lấy phần bù.

Ví dụ:

Giả sử
(a)₁₀ =3,
(b)₁₀ =5
khi đó
(c)₁₀ = a &
b cho ta
kết quả là
1:

```

0000.0000.0000.0011 a=3
& 0000.0000.0000.0101 (b)=5
0000.0000.0000.0001 c =1
c = a | b; cho ta kết quả là 7;
0000.0000.0000.0011 a =3
| 0000.0000.0000.0101 b =5
0000.0000.0000.0111 c =7
c = a ^ b; cho ta kết quả là 6;
0000.0000.0000.0011 a =3
^ 0000.0000.0000.0101 b=5
0000.0000.0000.0110 c =6
c = ~a; cho ta kết quả là 65532;
~ 0000.0000.0000.0011 a =3
1111.1111.1111.1100 c = 65532
c = a<<2; cho ta kết quả là (0000.0000.0000.1100) 2
c=a>>1; cho ta kết quả là (0000.0000.0000.0001) 2

```

- Toán tử chuyển đổi kiểu :

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

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

```

#include <stdio.h>

{

int a=3, b=5;

float c;
c= (float) a / (float) b;

printf("\n thương c = a / b =%6.2f", c);

return 0;

```

15

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT


}

- 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
-------------	-----------------

(), [], ->	Trái -> Phải
<div></div>	Phải -> Trái
- , ++ , -- , !	
* , / , %	Trái -> Phải
+ , -	Trái -> Phải
>> , <<	Trái -> Phải
< , <= , > , >= ,	Trái -> Phải
== , !=	Trái -> Phải
&	Trái -> Phải
^	Trái -> Phải
	Trái -> Phải
&&	Trái -> Phải

	Trái -> Phải
?:	Phải -> Trái
=, +=, -=, *=, /=, %=, &=, ^=, =, <<=, >>=	Phải -> Trái

2.5. Thủ tục vào và ra chuẩn

2.5.1 Vào ra ra bằng getchar(), putchar()

Cơ chế vào đơn giản nhất là đọc từng kí tự từ thiết bị vào chuẩn (bàn phím, màn hình) bằng getchar. Mỗi khi được gọi tới getchar() sẽ cho kí tự đọc vào tiếp theo. getchar cho giá trị EOF khi nó gặp cuối tệp trên bất cứ cái vào nào đang được đọc. Thư viện chuẩn định nghĩa hằng kí hiệu EOF là -1 (với #define trong tệp stdio.h) nhưng các phép kiểm tra phải viết dưới dạng EOF chứ không là -1 để cho độc lập với giá trị cụ thể.

Để đưa ra, putchar(c) sẽ đặt kí tự trên “thiết bị ra chuẩn”, cũng có giá trị mặc

định là màn hình.

Việc đưa ra cho printf cũng chuyển tới thiết bị ra chuẩn, các lời gọi tới putchar và printf có thể chen lẫn nhau.

Nhiều chương trình chỉ đọc trên một thiết bị vào và viết trên một thiết bị ra; với việc vào và ra của các chương trình thì sử dụng getchar, putchar kết hợp với printf là hoàn toàn thích hợp và đầy đủ. Điều này đặc biệt đúng khi làm việc với tệp và sử dụng công cụ đường ống nối đầu ra của chương trình này thành đầu vào của



chương trình tiếp. Chẳng hạn, xét chương trình lower, chuyển kí tự vào của nó thành chữ thường:

```
#include <stdio.h>
int main() /*chuyển kí tự vào thành chữ thường*/
{
    int c;
    while ((c = getchar()) != EOF)
        putchar(isupper(c) ? tolower(c) : c);
    return 0;
}
```

Các “hàm” isupper và tolower thực tế là các macro được xác định trong stdio.h, macro isupper kiểm tra xem đối của nó là chữ hoa hay không, cho giá trị khác 0 nếu đúng như vậy và cho 0 nếu nó là chữ thường. marco tolower chuyển chữ hoa thành chữ thường. Ta không cần biết tới cách các hàm này được cài đặt thế nào trên máy cụ thể, hành vi bên ngoài của chúng như nhau cho nên các chương trình sử dụng chúng không cần để ý tới tập kí tự.

Ngoài ra, trong thư viện vào/ ra chuẩn “các hàm” getchar và putchar là các macro và do vậy tránh được tốn phí về lời gọi hàm cho mỗi kí tự.

2.5.2 In ra theo khuôn dạng - Printf

Hai hàm printf để đưa ra và scanf để nhập vào cho phép chuyển ra các biểu diễn kí tự và số. Chúng cũng cho phép sinh ra hoặc thông dịch các dòng có khuôn dạng. Trong các chương trước, chúng ta đã dùng printf một cách hình thức mà chưa có những giải thích đầy đủ về nó. Bây giờ chúng ta sẽ mô tả đầy đủ và chính xác hơn cho hàm này.

printf(control, arg1, arg2,...)

printf chuyển, tạo khuôn dạng, và in các đối của nó ra thiết bị ra chuẩn dưới sự điều khiển của xâu control. Xâu điều khiển của control đều được đưa vào bằng kí tự % và kết thúc bởi một kí tự chuyển dạng. Giữa % và kí tự chuyển dạng có thể có. Dấu trừ (-), xác định việc dôn trái cho đối được chuyển dạng trong trường.

Xâu chữ số xác định chiều ngang tối thiểu của trường. Số được chuyển dạng sẽ được in ra trong trường tối thiểu với chiều ngang này, và sẽ rộng hơn nếu cần thiết. Nếu đối được chuyển có ít kí tự hơn là chiều ngang của trường thì nó sẽ được bổ sung thêm kí tự vào bên

trái (hoặc phải, nếu có cả chỉ báo dôn trái) để cho đủ chiều rộng trường. Kí tự bổ xung thêm sẽ là dấu trống thông thường hoặc số 0 nếu chiều ngang trường được xác định với số 0 đứng đầu.

Dấu chấm phân tách chiều ngang trường với xâu chữ số tiếp.

Xâu chữ số (độ chính xác) xác định ra số cực đại các kí tự cần phải in ra từ một xâu,

hoặc số các chữ số cần phải in ra ở bên phải dấu chấm thập phân của float hay double.

Bộ
thay
đổi



chiều dài l (chữ ell) chỉ ra rằng phần dữ liệu tương ứng là long chứ không phải là int.

Sau đây là các kí tự chuyển dạng và nghĩa của nó là:

d Đổi được chuyển sang kí pháp thập phân.

o Đổi được chuyển sang kí pháp hệ tám

x Đổi được chuyển sang cú pháp hệ mười sáu không dấu(không có 0x đứng trước). u

Đổi được chuyển sang kí pháp thập phân không dấu

c Đổi được coi là một kí tự riêng biệt.

s Đổi là chuỗi kí tự; các kí tự trong chuỗi được in cho tới khi gặp kí tự trống hoặc cho tới khi đủ số lượng kí tự được xác định bởi đặc tả về độ chính xác.

e Đổi được xem là float hoặc double và được chuyển sang kí pháp thập phân có dạng[-]m.nnnnnnE[+]xx với độ dài của chuỗi chứa n do độ chính xác xác định. Độ chính xác mặc định là 6.

f Đổi được xem là float hoặc double và được chuyển sang kí pháp thập phân có dạng [-]mmm.nnnnnn với độ dài của chuỗi các n do độ chính xác xác định. Độ chính xác mặc định

là 6. Lưu ý rằng độ chính xác không xác định ra số các chữ số có nghĩa phải in theo khuôn dạng f.

g Dùng %e hoặc %f, tùy theo loại nào ngắn hơn; không in các số không vô nghĩa.

Nếu kí tự đứng sau % không phải là kí tự chuyển dạng thì kí tự đó sẽ được in ra; vậy % sẽ được in ra bởi %%.

Phần lớn các chuyển dạng là hiển nhiên, và đã được minh họa ở các chương trước. Một biệt lệ là độ chính xác liên quan tới các xâu. Bảng sau đây sẽ chỉ ra hiệu quả của các loại đặc tả trong việc in “hello, world” (12 kí tự). Chúng ta đặt dấu hai chấm xung quanh chuỗi kí tự in ra để có thể thấy sự trải rộng của nó

:%10s: :hello, world:

:%-10s: :hello, world:

:%20s: :hello, world :

:%-20s: : hello, world:

:%20.10s: :hello, world:

18

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

:%-20.10s: :hello, word:

:%.10s :hello,word:

Lưu ý: printf dùng đối thứ nhất của nó để quyết định xem có bao nhiêu đối theo sau và kiểu của chúng là gì. Hàm sẽ bị lẫn lộn và ta sẽ nhận được câu trả lời vô nghĩa nếu không đủ số đối hoặc đối có kiểu sai.

2.5.3 Nhập vào có khuôn dạng - scanf

Hàm
scanf
là
hàm



tương tự printf, đưa ra nhiều chức năng chuyển dạng như của printf nhưng theo chiều ngược lại.

`scanf(control, arg1, arg2,...)`

scanf đọc các kí tự từ thiết bị vào chuẩn, thông dịch chúng tương ứng theo khuôn dạng được xác định trong control, rồi lưu trữ kết quả trong các đối còn lại. Đối điều khiển sẽ được mô tả sau đây; các đối khác đều phải là con trỏ để chỉ ra nơi dữ liệu chuyển dạng tương ứng cần được lưu trữ.

Xâu điều khiển thường chứa các đặc tả chuyển dạng, được dùng để thông dịch trực tiếp dãy vào. Xâu điều khiển có thể chứa:

Dấu cách, dấu tab hoặc dấu xuống dòng (“các kí tự khoảng trắng”), thường bị bỏ qua. Các kí tự thông thường (khác%) được xem là ứng với kí tự khác khoảng trắng trong dòng vào.

Các đặc tả chuyển dạng, bao gồm kí tự %, kí tự cắt bỏ gán *(tùy chọn), một số tùy chọn xác định ra chiều ngang cực đại của trường, và một kí tự chuyển dạng. Đặc tả chuyển dạng điều khiển việc chuyển dạng cho trường vào tiếp theo. Thông thường kết quả sẽ được đặt vào biến được trỏ tới bởi đối tương ứng. Tuy nhiên, nếu việc cắt bỏ gán được nêu ra bởi kí tự * thì trường vào sẽ bị bỏ qua. Trường vào được xác định như một xâu các kí tự khác khoảng trắng và được kéo dài hoặc tới kí tự khoảng trắng tiếp hoặc cho tới khi chiều ngang của trường.

Kí tự chuyển dạng chỉ ra việc thông dịch cho trường vào; đối tương xứng phải là một con trỏ theo yêu cầu của lời gọi bởi ngữ nghĩa giá trị của C. Các kí tự chuyển dạng sau đây là hợp pháp:

d nhận một số nguyên trong trường vào; đối tương ứng phải là con trỏ nguyên.

o nhận số nguyên hệ tám trong trường vào (có hoặc không có số không đứng trước) đối tương ứng phải là con trỏ nguyên.

x nhận số nguyên hệ mười sáu trong vào (có hoặc không có 0x đứng trước); đối tương ứng phải là con trỏ nguyên.

h nhận số nguyên short trong trường vào; đối tương ứng phải là con trỏ nguyên short.

c nhận một kí tự; đối tương ứng phải là con trỏ kí tự; kí tự vào tiếp được đặt vào chỗ chỉ ra. Trong trường hợp này không bỏ qua các kí tự khoảng trắng; để đọc kí tự khác khoảng trắng tiếp tục dùng %1s.

s nhận một xâu kí tự; đối tương ứng phải là con trỏ kí tự trỏ tới bằng các kí tự đủ lớn

để nhận được xâu và dấu kết thúc \0 sẽ được thêm vào. Xâu kí tự được nhập qua hàm scanf sẽ không nhận khoảng trắng, tab.

f nhận số dấu phẩy động; đối tương ứng phải là con trỏ tới float. Kí tự chuyển dạng e đồng nghĩa với f. Khuôn dạng vào cho float là một dấu tùy chọn, một xâu các số có thể chứa dấu chấm thập phân và một trường mũ tùy chọn chứa E hoặc e theo sau là một số nguyên có dấu.

Có thể viết thêm l (chữ ell) vào trước kí tự



chuyển dạng d, o và x để chỉ ra rằng con trỏ tới long chứ không phải là int sẽ xuất hiện trong danh sách đối. Tương tự, có thể đặt l vào trước các kí tự chuyển dạng e hoặc f để chỉ ra rằng con trỏ tới double chứ không tới float trong danh sách đối.

Chẳng hạn, lời gọi

```
int i;
float x;

char name[50];
scanf ("%d %f %s", &i, &x, name);
```

Với dòng vào

25 54.32 E-1 Thompson

Sẽ gán giá trị 25 cho i, giá trị 5.432 cho x và xâu “Thompson” có cả dấu kết thúc \0, cho name. Ba trường vào có thể cách nhau bởi nhiều dấu cách, dấu tab và dấu xuống dòng.

Lời gọi

```
int i;
float x;
char name[50];
```

```
scanf("%2d %f %*d %2s", &i, &x, name);
```

Với đầu vào

56_789 0123 45a72

Sẽ gán 56 cho i, gán 789.0 cho x, nhảy qua 0123 và đặt xâu "45" vào name. Lời gọi tiếp tới bất kì trình vào nào cũng sẽ bắt đầu tìm tới chữ a. Trong hai ví dụ trên, name là con trỏ và do vậy phải không được đứng sau &.

Xét ví dụ khác, chương trình bàn tính thô sơ có thể được viết với scanf để thực hiện chuyển dạng cái vào

```
#include <stdio.h>
int main() /*bàn tính thô sơ*/
{
    double sum, v;
    sum = 0;
    while(scanf("%lf", &v) != EOF)
```

20

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```
    printf("\ t%.2f \ n", sum + = v);
    return 0;
}
```

scanf dừng lại khi nó vét hết xâu điều khiển hoặc khi dữ liệu vào nào đó không sánh đúng với đặc tả điều khiển. Hàm này cho giá trị là số các khoản mục vào đã được đối sánh đúng và được gán. Do vậy có thể dùng giá trị của hàm để xác định xem đã tìm thấy bao nhiêu dữ

liệu
vào.
Khi
gặp
dấu
hiệu
cuối
tệp
hàm
sẽ
cho
giá
trị
EOF,
lưu ý
rằng
giá
trị



này khác 0, giá trị 0 có nghĩa là kí tự vào tiếp sẽ không đối sánh đúng với đặc tả đầu tiên trong xâu điều khiển. Lời gọi tiếp tới scanf sẽ tìm đọc tiếp ngay sau kí tự cuối cùng vừa cho.

Điều cần lưu ý nhất là: các đối của scanf phải là con trỏ. Lỗi hay mắc nhất là viết scanf ("%d", n); Đúng ra phải là scanf ("%d", &n);

Tương tự như các hàm scanf và printf là sscanf và sprintf, thực hiện các phép chuyển dạng tương ứng nhưng làm việc trên các xâu chứ không trên tệp. Khuôn dạng tổng quát của chúng là:

```
sprintf(string, control, arg1, arg2,...)
sscanf(string, control, arg1, arg2,...)
```

sprintf tạo khuôn dạng cho các đối trong arg1, arg2, v.v... tương ứng theo control như trước, nhưng đặt kết quả vào string chứ không đưa ra thiết bị chuẩn. Tất nhiên string phải đủ lớn để nhận kết quả. Ví dụ: nếu name là một mảng kí tự và n là nguyên thì

```
sprintf (name, "temp%d", n);
```

Tạo ra một xâu có dạng tempnnn trong name với nnn là giá trị của n.

scanf làm công việc ngược lại - nó nhòm vào string tương ứng theo khuôn dạng trong control và đặt các giá trị kết quả vào arg1, arg2... Các đối phải là con trỏ. Lời gọi

```
sscanf(name, "temp%d", n);
```

đặt cho n giá trị của xâu các chữ số đứng sau temp trong name.

2.5.4 Thâm nhập vào thư viện chuẩn

Mỗi tệp gốc có tham trỏ tới hàm thư viện chuẩn đều phải chứa dòng khai

báo #include < tên_tệp_thư_viện >

ở đầu tệp văn bản chương trình. Dấu ngoặc nhọn < tên_tệp_thư_viện > thay cho dấu nháy thông thường để chỉ thị cho trình biên dịch tìm kiếm tệp trong danh mục chứa thông tin tiêu đề chuẩn được lưu trữ trong thư mục include. Trong trường hợp chúng ta sử dụng kí tự "tên_tệp_thư_viện" trình biên dịch sẽ tìm kiếm tệp thư viện tại thư mục hiện tại.

Chỉ thị #include "tên_tệp_thư_viện" còn có nhiệm vụ chèn thư viện hàm của người sử dụng vào vị trí của khai báo. Trong ví dụ sau, chúng ta sẽ viết chương trình thành 3 tệp, tệp define.h dùng để khai báo tất cả các hằng sử dụng trong chương trình, tệp songuyen.h dùng khai báo nguyên mẫu của hàm và mô tả chi tiết các hàm giống như một thư viện của người sử dụng, tệp mainprg.c là chương trình chính ở đó có những lời gọi hàm từ tệp songuyen.h.

Ví dụ: Xây dựng một thư viện đơn giản mô tả tập thao tác với số nguyên bao gồm: tính tổng, hiệu, tích, thương, phần dư, ước số chung lớn nhất của hai số nguyên dương a, b.

```
/* Nội dung tệp define.h */
#include <stdio.h>
#include <conio.h>
```

```
#include <dos.h>
#define F1 59
```



```
#define F2 60
#define F3 61
#define F4 62
#define F5 63
#define F6 64
#define F1 65
#define F10 68

/* Nội dung tệp songuyen.h */

void Init_Int( int *, int *);
int Tong_Int(int , int );
int Hieu_Int(int, int );
int Tich_Int(int, int );
float Thuong(int , int );
int Mod_Int(int, int);
int UCLN(int , int);
void Init_Int( int *a, int *b ){
```

```

        printf("\n Nhập a = "); scanf( "%d", a);
        printf("\n Nhập b = "); scanf( "%d", b);
    }
    int Tong_Int( int a, int b ){
        printf("\n Tổng a + b =%d", a + b);
        delay(2000);
        return(a+b);
    }
    int Hieu_Int( int a, int b ){
        printf("\n Hiệu a - b =%d", a - b);
        delay(2000);
        return(a - b);
    }
    int Tich_Int( int a, int b ){
        printf("\n Tích a * b =%d", a * b);
        delay(2000);
        return(a*b);
    }
}

```

```

float Thuong_Int( int a, int b ){
    printf("\n Thương a / b =%6.2f", (float) a /(float)
    b); delay(2000);
    return((float) a /(float) b);
}
int Mod_Int( int a, int b ){
    printf("\n Phần dư a % b =%d", a % b);
    delay(2000);
}

```




```
        return(a%b);
    }
    int UCLN( int a, int b) {
        while (a != b) {
            if ( a > b) a -=b;
            else b-=a;
        }
        printf("\n UCLN =%d", a);
        delay(2000); return(a);
    }

    /* Nội dung tệp mainprg.c */
    #include "define.h"
    #include "songuyen.c"
    void thuchien(void){
        int a, b, control = 0; char c; textmode(0);
        do {
            clrscr();
            printf("\n Tập thao tác với số nguyên");
```

```

printf("\n F1- Nhập hai số nguyên");
printf("\n F2- Tổng hai số nguyên");
printf("\n F3- Hiệu hai số nguyên");
printf("\n F4- Tích hai số nguyên");
printf("\n F5- Thương hai số nguyên");
printf("\n F6- Phần dư hai số nguyên");
printf("\n F7- UCLN hai số nguyên");
printf("\n F10- Trở về");
c = return 0;
switch(c) {
    case F1: Init_Int(&a, &b); control =1; break;
    case F2:
        if (control) Tong_Int(a, b);
        break;
    case F3:
        if (control) Hieu_Int(a, b);

```

23

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```

        break;
    case F4:
        if (control) Tich_Int(a, b);
        break;
    case F5:
        if (control) Thuong_Int(a, b);
        break;
    case F6:
        if
        (control)
        Mod_Int(a,
        b);
        break;
    case F7:
        if
        (control)
        UCLN_Int(a,
        b);
        break;
    }
} while (c!=F10);
}
int main() {
    thuchien();
    return 0;
}

```



3. CÁC CẤU TRÚC LỆNH ĐIỀU KHIỂN

3.1. Câu lệnh khối

Tập các câu lệnh được bao bởi hai dấu `{ . . . }` được gọi là một câu lệnh khối. Về cú pháp, ta có thể đặt câu lệnh khối ở một vị trí bất kì trong chương trình. Tuy nhiên, nên đặt các câu lệnh khối ứng với các chu trình điều khiển lệnh như `for`, `while`, `do . . while`, `if . . else`, `switch` để hiển thị rõ cấu trúc của chương trình.

Ví
dụ:

```
if (a  
> b )  
{
```



```
câu_lệnh;
```

```
}
```

3.2. Cấu trúc lệnh if

Dạng 1:

```
if ( biểu thức)
```

```
    câu_lệnh;
```

Nếu biểu thức có giá trị đúng thì thực hiện câu_lệnh; Câu lệnh có thể hiểu là câu lệnh đơn hoặc câu lệnh khối, nếu câu lệnh là lệnh khối thì nó phải được bao trong { . . }. **Dạng 2:**

```
if (biểu_thức)
```

```
    câu_lệnh_A;
```

```
else
```

```
    câu_lệnh_B;
```

Nếu biểu thức có giá trị đúng thì câu_lệnh_A sẽ được thực hiện, nếu biểu thức có giá trị sai thì câu_lệnh_B sẽ được thực hiện.

Dạng 3: Được sử dụng khi có nhiều lệnh if lồng nhau hoặc phải kiểm tra nhiều biểu thức khác nhau.

```
if (biểu_thức_1)
```

```
    câu_lệnh_1;
```

```
else if (biểu_thức_2)
```

```
    câu_lệnh_2;
```

```
.....
```

```
else if (biểu_thức_k)
```

```
    Câu_lệnh_k;
```

```
else
```

```
    câu_lệnh_k+1;
```

25

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Nếu biểu_thức_i có giá trị đúng ($0 < i \leq k$) thì câu_lệnh_i sẽ được thực hiện, nếu không biểu thức nào có giá trị đúng thì câu_lệnh_k+1 sẽ được thực hiện.

Ví dụ: Tìm số lớn nhất trong hai số a và b:

```
#include <stdio.h>
```

```
int main() {
```

```
    float a, b, max;
```

```
    printf("\n Nhập a="); scanf("%f", &a); /* nhập giá trị
```

```
cho
```

```

    biến a*/
    printf("\n Nhập b="); scanf("%f", &b); /* nhập giá trị
    cho biến b*/
    if (a>b) max=a;
    else max= b;
    printf("\n Max(a,b)=%6.2f",max);
    return 0;
}

```

Ghi chú:

Toán tử: &(tên_biến) lấy địa chỉ của biến. Câu lệnh scanf("%f",&a) có nghĩa là nhập một số thực vào địa chỉ ô nhớ dành cho biến a.

Ví dụ: Viết chương trình giải phương trình bậc 2 : $ax^2 + bx + c = 0$

```

#include <stdio.h>
#include <math.h>
int main(){
    float a, b, c, x1, x2, delta;
    printf("\n Giải phương trình bậc 2:");
    /*đọc các hệ số a, b, c từ bàn phím*/
    printf("\n Nhập hệ số a="); scanf("%f",&a);
    printf("\n Nhập hệ số b="); scanf("%f",&b);
    printf("\n Nhập hệ số c="); scanf("%f",&c);
    /* tính delta = b2 - 4ac*/
    delta=b*b-4*a*c;
    if (delta==0){
        printf("\n phương trình có 2 nghiệm kép x1=x2=%f",
        -b/(2*a));
    }
    else if(delta>0){
        printf("\n Phương trình có hai nghiệm");
        x1= ( -b + sqrt(delta) ) / (2*a);
        x2= ( -b - sqrt(delta) ) / (2*a);
        printf(" x1 = % 6.2f x2=%6.2f", x1,x2);
    }
    else {

```

```

        printf("\n Phương trình có nghiệm phức:");
        x1 = -b / (2 a); /* phần thực */
        x2 = ( sqrt( -delta) ) / (2 * a);
        printf(" Real = %6.2f Im = % 6.2f", x1, x2);
    }
    return 0;

```

}

3.3. Cấu trúc lệnh



switch

Cấu trúc lệnh if thực hiện một phương án đúng trong hai phương án có thể xảy ra. Cấu trúc lệnh **switch** dùng để lựa chọn và thực hiện các phương án đúng có thể xảy ra. Cú pháp

```
switch(biểu_thức_nguyên){  
    case hằng_nguyên_1: câu_lệnh_1; break;  
    case hằng_nguyên_2: câu_lệnh_2; break;  
    case hằng_nguyên_3: câu_lệnh_3; break;  
    .....  
    case hằng_nguyên_n: câu_lệnh_n; break;  
    default: câu_lệnh_n+1; break;  
}
```

Thực hiện: Nếu biểu_thức_nguyên có giá trị trùng với hằng_nguyên_i thì câu_lệnh_i trở đi sẽ được thực hiện cho tới khi nào gặp từ khoá break để thoát khỏi chu trình. **Ví dụ:** Nhập một số nguyên dương từ bàn phím và xác định xem số nguyên đó có phải là các số từ 1..10 hay không? Trong trường hợp không phải là các số nguyên từ 1..10 hãy đưa ra thông báo "số lớn hơn 10".

```
#include <stdio.h>
```

```

int main(){
    int n;
    printf("\n Nhập n=");scanf("%d",&n);
    switch(n){
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
        case 10: printf("\n Số từ 1. .10"); break;
        default : printf("\n Số lớn hơn 10"); break;
    }

```

```

        return 0;
    }

```

3.4. *Vòng lặp for*

Cú pháp:

```

for(biểu_thức_1; biểu_thức_2; biểu_thức_3)
    Câu_lệnh;

```



Câu_lệnh: Có thể là lệnh đơn hoặc lệnh khối, nếu là lệnh đơn thì câu lệnh trong thân chu trình for không cần thiết phải bao trong hai kí hiệu {, }. Nếu là lệnh khối (thân chu trình for có hơn một lệnh) thì nó phải được bao trong hai kí hiệu {, }.

Thực hiện:

Biểu_thức_1: Được gọi là biểu thức khởi đầu có nhiệm vụ khởi đầu các biến sử dụng trong chu trình, biểu_thức_1 chỉ được thực hiện duy nhất một lần khi bắt đầu bước vào chu trình. Nếu trong chu trình phải khởi đầu nhiều biểu thức thì mỗi biểu thức được phân biệt với nhau bởi một kí tự ' '.

Biểu_thức_2: Được gọi là biểu thức kiểm tra và được thực hiện ngay sau khi thực hiện xong biểu_thức_1, nếu biểu thức kiểm tra có giá trị đúng (khác 0) thì câu lệnh trong thân của chu trình for sẽ được thực hiện, nếu biểu thức kiểm tra có giá trị sai thì điều khiển của chương trình chuyển về lệnh kế tiếp ngay sau thân của chu trình for.

Biểu_thức_3: Được gọi là biểu thức khởi đầu lại có nhiệm vụ khởi đầu lại các biến trong chu trình và được thực hiện ngay sau khi thực hiện xong câu_lệnh. Chu trình sẽ được lặp lại bằng việc thực hiện biểu thức kiểm tra.

Ví dụ: Viết chương trình in ra màn hình dãy các kí tự theo dạng sau:

A B C D . . . Z


```
a b c d . . . z
Z Y X W . . . A
z y X w . . . a
```

```
/* chương trình in dãy kí tự */
#include <stdio.h>
int main() {
    char ch;
    for(ch = 'A'; ch <= 'Z'; ch++)
        printf("%3c", ch);
    printf("\n");
    for(ch = 'a'; ch <= 'z'; ch++)
        printf("%3c", ch);
    printf("\n");
    for(ch = 'Z'; ch >= 'A'; ch--)
        printf("%3c", ch);
```

28

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```
printf("\n");
for(ch = 'z'; ch >= 'a'; ch--)
    printf("%3c", ch);
printf("\n");
return 0;
}
```



Ghi

chú:

Đối
với
ngôn
ngữ
C,
kiểu
dữ
liệu
char
thực
chất
là
một
số

nguyên có kích cỡ 1 byte, giá trị của byte là vị trí của kí tự trong bảng mã ASCII. Do vậy, chương trình trên có thể viết lại bằng cách sau:

Ví dụ:

```
/* chương trình in dãy kí tự */
#include <stdio.h>
int main() {

    int ch;
    for(ch =65; ch<=90; ch++)
        printf("%3c",ch);
    printf("\n");
    for(ch =97; ch<=122; ch++)
        printf("%3c",ch);
    printf("\n");
    for(ch ='Z'; ch>='A'; ch--)
        printf("%3c",ch);
    printf("\n");
    for(ch ='z'; ch>='a'; ch--)
        printf("%3c",ch);
    printf("\n");return 0;

}
```

Ví dụ: Viết chương trình giải bài toán cổ "Trăm trâu trăm cỏ".

```
#include <stdio.h>

int main() {
    unsigned int x, y, z; /* khai báo số trâu đứng, trâu nằm,
    trâu già*/
    for( x=0;x<=20;x++){
        for(y=0;y<=33;y++){
            for(z=0;z<100;z+=3){
                if(x + y + z ==100 && (5*x + 3 *y + ( z / 3))==100){
                    printf("\n Trâu đứng:%5d",x);
```

```
                printf(" Trâu nằm:%5d ",y);
                printf(" Trâu già:%5d", z);
            }
        }
    }
}
```

```
return 0;
```

```
}
```

3.5. *Vòng lặp*



không xác định while

Cú pháp:

```
while(biểu_thức)
    câu_lệnh;
```

Trong khi biểu thức còn đúng thì câu lệnh sẽ được thực hiện, nếu biểu thức có giá trị sai điều khiển của chương trình chuyển về lệnh kế tiếp ngay sau thân của while. Nếu trong thân của while có nhiều hơn một lệnh thì nó phải được bao trong hai kí tự { . }. **Ví dụ:** Đếm số chữ số, khoảng trắng (space), dấu tab, dấu về đầu dòng và những kí tự khác được nhập từ bàn phím.

```
#include <conio.h>
#include <stdio.h>
#define ESC 27 /* mã của phím ESC */
#define ENTER 13
int main() {
    int number=0, space=0, tab=0, enter=0, other=0;
    char ch;
```

```

while( ( ch=getch() ) != ESC){ /* thực hiện nếu không
phải là ESC*/
    if(ch>='0' && ch <='9')
        number++;
    else if(ch == ' ') space++;
    else if(ch == '\t') tab++;
    else if(ch == ENTER) enter ++;
    else other++;
}
printf("\n Số chữ số là: %d", number);
printf("\n Số dấu trống là: %d", space);
printf("\n Số dấu tab là: %d", tab);
printf("\n Số dấu xuống dòng là: %d", enter);
printf("\n Các kí tự khác: %d", other);

return 0;

```

30

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```

}

```

Ví dụ: Tìm tổng $S = 1 + 1/3 + 1/5 + \dots + 1/(2n-1)$ với độ chính xác ϵ ($1/n \geq \epsilon$);

```

#include <stdio.h>
#include <conio.h>
int main(){
    int i =1;
    float s=0, epsilon;

```

```

clrscr();

```



```

    printf("\n Nhập độ chính xác epsilon=");
    scanf("%f",&epsilon);
    while( ( (float) 1 / (float i) )>=epsilon) {
        s+=(float) 1 / (float) i;
        i+=2;
    }
    printf("\n Tổng s=%6.2f", s);
    return 0;
}

```

Ví dụ: Tính e^x theo công thức xấp xỉ chuỗi Taylor với $e = x^n/n!$.

$$e^x = 1 + x/1! + x^2/2! + x^3/3! + \dots + x^n/n!$$

```

#include <stdio.h>
#include <conio.h>
int main(){
    float e_mu_x, epsilon, x, t;
    int n; clrscr();
    printf("\n Nhập x="); scanf("%f", &x);
    printf("\n Nhập độ chính xác epsilon="); scanf("%f",
&epsilon);
    e_mu_x = 1; n = 1; t = x;
    while ( t >=epsilon) {
        e_mu_x += t ;
        n++; t = t * (x/n);
    }
    printf("\n e mũ %6.3f = %6.3f", x, e_mu_x);
    return 0;
}

```

3.6. Vòng lặp không xác định do... while

Cú pháp:

```

do {
    câu_lệnh;
} while(biểu_thức);

```

Thực hiện câu lệnh trong khi biểu_thức vẫn còn đúng, nếu biểu thức có giá trị sai, điều khiển chương trình chuyển về lệnh kế tiếp ngay sau while(biểu_thức). **Ví dụ:** Viết chương trình xây dựng tập thao tác cộng, trừ, nhân, chia, lấy phần dư của hai số nguyên a,b.

```

#include <stdio.h>
#include <dos.h> /* sử dụng hàm delay() */
#define F1 59 /* định nghĩa phím F1 */

```



```
#define F2 60 /* định nghĩa phím F2 */
#define F3 61 /* định nghĩa phím F3 */
#define F4 62 /* định nghĩa phím F4 */
#define F5 63 /* định nghĩa phím F5 */
#define F6 64 /* định nghĩa phím F6 */
#define F10 68 /* định nghĩa phím F10 */
int main(){
    int a, b, control=0; char key;
    clrscr();
    do {
        printf("\n Tập thao tác với hai số nguyên a,
        b"); printf("\n F1- Nhập hai số nguyên a,b");
        printf("\n F2-Tổng hai số nguyên");
        printf("\n F3-Hiệu hai số nguyên");
        printf("\n F4- Tích hai số nguyên");
        printf("\n F5- Thương hai số nguyên");
        printf("\n F6- Modul hai số nguyên");
        printf("\n F10- Trở về");
```

```

key = return 0;
switch(key) {
    case F1:
        printf("\n Nhập a="); scanf("%d", &a);
        printf("\n Nhập b="); scanf("%d", &b);
        control =1; break;
    case F2:
        if( control !=0 )
            printf("\n Tổng a + b =%d", a+b);
        break;
    case F3:
        if( control !=0 )
            printf("\n Hiệu a - b =%d", a - b);
        break;
    case F4:
        if( control !=0 )
            printf("\n Tích a * b =%d", a * b);

```

32

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```

        break;
    case F5:
        if( control !=0 )
            printf("\nThương a*b=%6.2f", (float)a/
(float)b);
        break;
}

```

}



```

while (key!=F10) ;

return 0;

}

```

3.7. Lệnh break và lệnh Continue

Lệnh break cho phép ra khỏi các chu trình với các toán tử for, while và switch.

Ví dụ 1: Tính tổng $S = 1+2+3+4+5$

```

s=0;
i=1;
for (i=1; i<=10; i++)
{ if (i==6) break;
  s=s+i;
}

```

Lệnh continue dùng để bắt đầu một vòng mới của chu trình chứa nó. Lệnh continue trong trường hợp for điều khiển được chuyển về bước khởi đầu lại. Còn trong while, do...while quyền điều khiển chương trình được chuyển đến bước kiểm tra điều kiện.

Ví dụ 1: Tính tổng $S = 1+2+3+4+5+7+8+9+10$

```

s=0;
i=1;
for (i=1; i<=10; i++)
{ if (i==6) continue;
  s=s+i;
}

```

Ví dụ 1: $S = 1+2+3+4+5 = 15$

Ví dụ 2: $S = 1+2+3+4+5+7+8+9+10 = 49$

3.8. Bài tập cuối chương

1. 43. BASIC_CONDITIONAL_106. Kiểm tra một số có chia hết cho hai số khác hay không

Viết chương trình C cho phép nhập vào 3 số a, b, c. Thực hiện kiểm tra a có chia



hết cho
b và c
hay
không.
Nếu a
chia hết
cho b
và c thì
in ra 1,
ngược
lại in ra
0.

INPUT
15 3 5
OUTPUT
1

2. 44.

BASIC_CONDITIONAL_107. Kiểm tra ba góc của một tam giác

Ba góc của một tam giác sẽ thỏa mãn là số nguyên dương và có tổng bằng 180. Viết chương trình C cho phép nhập vào 3 giá trị và kiểm tra xem có thỏa mãn là ba góc của một tam giác hay không. Nếu đúng in ra 1, ngược lại in ra 0. INPUT

30 60 90

OUTPUT

1

3. 45. BASIC_CONDITIONAL_108. Kiểm tra ba cạnh của một tam giác

Ba cạnh được gọi là cạnh của một tam giác nếu chúng nguyên dương và tổng của hai cạnh luôn lớn hơn cạnh còn lại.

Viết chương trình C cho phép nhập vào 3 cạnh và kiểm tra xem có thỏa mãn là 3 cạnh của một tam giác hay không. Nếu đúng in ra 1, sai in ra 0. INPUT

7 4 10

OUTPUT

1

4. 46. BASIC_CONDITIONAL_109. Kiểm tra tam giác (vuông, cân, đều)

Viết chương trình C cho phép nhập vào ba cạnh của một tam giác và kiểm tra xem ba cạnh có thỏa mãn là ba cạnh của tam giác đặc biệt nào đó, như là tam giác vuông, cân, đều hay không.

Nếu là tam giác đều in ra 3

Nếu là tam giác cân in ra 2

Nếu là tam giác vuông in ra 1

INPUT

34

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

3 3 3

OUTPUT

3

5. 47. BASIC_CONDITIONAL_110. Phương trình bậc hai

Phương trình bậc 2 là phương trình dạng $ax^2 + bx + c = 0$.

Viết



chương trình C cho phép nhập vào a,b,c và thực hiện giải phương trình bậc 2.
Nếu vô nghiệm thì in ra dòng NO
Nếu có nghiệm thì in các nghiệm (luôn lấy 2 chữ số thập phân sau dấu chấm phẩy) cách nhau một khoảng trắng.

INPUT

8 -4 -2

OUTPUT

0.81 -0.31

6. 48. *BASIC_CONDITIONAL_952. Tìm giá trị nhỏ nhất trong ba số*

Viết chương trình C cho phép nhập vào ba số và tìm giá trị nhỏ nhất trong ba số đó.

INPUT

10 20 30

OUTPUT

10

7. 49. *BASIC_DEFAULT_27. Đếm chữ số chẵn và chữ số lẻ*

Nhập một số nguyên dương N không quá 9 chữ số. Hãy đếm xem N có bao nhiêu chữ số lẻ và bao nhiêu chữ số chẵn. Nếu không tồn tại số lẻ hoặc số chẵn thì in ra kết quả là 0 cho loại số tương ứng

INPUT

12345678

OUTPUT

4 4

8. 50. *BASIC_DEFAULT_91. Tìm số đẹp (số nguyên tố và có tổng các chữ số là một số trong dãy fibonacci)*

Một số được coi là đẹp nếu nó là số nguyên tố và tổng chữ số là một số trong dãy Fibonacci. Viết chương trình liệt kê trong một đoạn giữa hai số nguyên cho trước có bao nhiêu số đẹp như vậy

INPUT

2 50

OUTPUT

2 3 5 11 17 23 41

9. 51. *BASIC_DEFAULT_92. Tìm số đẹp (số thuận nghịch lệch phát)*

Một số
được
coi là
số đẹp
nếu nó
là số
thuận



nghịch, có chứa ít nhất một chữ số 6, và tổng các chữ số của nó có chữ số cuối cùng là 8. Viết chương trình liệt kê trong một đoạn giữa hai số nguyên cho trước có bao nhiêu số đẹp như vậy INPUT

1 500

OUTPUT

161

10. 52. BASIC_DEFAULT_328. Phân tích một số thành các thừa số nguyên tố (x)

Viết chương trình phân tích một số nguyên thành các thừa số nguyên tố.

INPUT

28

OUTPUT

2x2x7

11. 53. BASIC_LOOP_111. Tìm tổng các số tự nhiên nằm trong khoảng a, b

Viết chương trình C cho phép nhập vào hai số a, b. Thực hiện tính tổng các số tự nhiên nằm trong khoảng a, b và in ra màn hình. (Lưu ý người dùng có thể nhập a lớn hơn b)

INPUT

1 10

OUTPUT

55

12. 54. BASIC_LOOP_112. Bảng cửu chương

Viết chương trình C cho phép người dùng nhập vào n (với n nằm trong khoảng từ 1-9), thực hiện in ra bảng cửu chương của n. (mỗi phần tử cách nhau một khoảng trắng)

INPUT

5

OUTPUT

36

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

5 10 15 20 25 30 35 40 45 50

13. 55. BASIC_LOOP_113. Đếm số chữ số của một số bất kỳ

Viết chương trình C cho phép nhập vào một số n và kiểm tra xem n có bao nhiêu chữ số.

INPUT

1234



OUTPUT

4

14. 56. BASIC_LOOP_114. Tìm chữ số đầu tiên và cuối cùng của một số bất kỳ

Viết chương trình C cho phép nhập vào một số n bất kỳ, tìm và in ra chữ số đầu tiên và cuối cùng của n.

INPUT

1234

OUTPUT

1 4

15. 57. BASIC_LOOP_115. Đổi chỗ chữ số đầu tiên và chữ số cuối cùng của một số

Viết chương trình C cho phép nhập vào một số nguyên n và thực hiện đổi vị trí của chữ cái đầu tiên và chữ cái cuối cùng.

Lưu ý trong trường hợp chữ số cuối cùng là 0 thì khi đổi chỗ sẽ được loại bỏ (ví dụ 9800 -> 809)

INPUT

1234

OUTPUT

4231

16. 58. BASIC_LOOP_116. Tìm tổng các chữ số của một số

Viết chương trình C cho phép nhập vào một số n, thực hiện tìm tổng các chữ số của n và in ra màn hình.

INPUT

1234

OUTPUT

10

17. 59. BASIC_LOOP_117. Tìm tích của các chữ số của một số

Viết chương trình C cho phép nhập vào một số n, thực hiện tìm tích của các chữ số của n và in ra màn hình.

INPUT

1234

OUTPUT



BASIC_LOOP_118. Tìm số đảo ngược của một số

Viết chương trình C cho phép nhập vào một số n và in ra số đảo ngược của n.
Lưu ý nếu chữ số cuối cùng là 0 thì khi in số đảo ngược có thể loại bỏ số này

INPUT

1234

OUTPUT

4321

19. 61. BASIC_LOOP_121. Liệt kê tất cả các ước của một số cho trước

Viết chương trình C cho phép nhập vào một số n và in ra tất cả các ước số của n.
(Mỗi ước được liệt kê cách nhau một khoảng trắng)

INPUT

12

OUTPUT

1 2 3 4 6 12

20. 62. BASIC_LOOP_122. Tìm giai thừa của một số

Viết chương trình C cho phép nhập một số tự nhiên n và tính giai thừa của n.

INPUT

5

OUTPUT

120

21. 63. BASIC_LOOP_123. Tìm ước chung lớn nhất của hai số

Viết chương trình C cho phép nhập vào hai số a, b. Thực hiện tìm ước chung lớn nhất của a và b và in ra màn hình.

INPUT

12 30

OUTPUT

38

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

6

22. 64. BASIC_LOOP_124. Tìm bội chung nhỏ nhất của hai số

Viết chương trình C cho phép nhập vào hai số a và b. Thực hiện tìm bội chung nhỏ nhất của a và b và in ra màn hình.

INPUT

12 30



OUTPUT

60

23. 65. *BASIC_LOOP_125. Kiểm tra một số có phải là số nguyên tố*

Viết chương trình C cho phép nhập vào 1 số và kiểm tra xem số đó có phải là số nguyên tố hay không. Nếu đúng in ra 1, sai in ra 0.

INPUT

11

OUTPUT

1

24. 66. *BASIC_LOOP_126. Liệt kê các số nguyên tố nhỏ hơn n*

Viết chương trình C cho phép nhập vào số n và liệt kê tất cả các số nguyên tố nhỏ hơn n. (Các số nguyên tố in ra cách nhau một khoảng trắng)

INPUT

20

OUTPUT

2 3 5 7 11 13 17 19

25. 67. *BASIC_LOOP_127. Tính tổng các số nguyên tố nhỏ hơn n*

Viết chương trình C cho phép nhập vào n và tính tổng các số nguyên tố nhỏ hơn n.

INPUT

10

OUTPUT

17

26. 68. *BASIC_LOOP_129. Phân tích một số thành các thừa số nguyên tố*

Viết chương trình C cho phép nhập vào một số và phân tích thành thừa số các số nguyên tố. (Mỗi thừa số nguyên tố cách nhau một khoảng trắng, mỗi thừa số nguyên tố chỉ liệt kê một lần)

INPUT

10

OUTPUT

2 5

27.
69.



BASIC_LOOP_130. Kiểm tra một số có phải là số armstrong hay không

Số armstrong là số A có n chữ số và thỏa mãn tổng của lũy thừa bậc n của từng chữ số trong A bằng chính nó. Ví dụ: $371 = 3^3 + 7^3 + 1^3$ Viết chương trình C kiểm tra một số xem có phải là số armstrong hay không. Nếu đúng in ra 1, sai in ra 0.

INPUT

371

OUTPUT

1

28. 70. BASIC_LOOP_131. Liệt kê các số armstrong nhỏ hơn n

Viết chương trình C cho phép nhập vào n và thực hiện liệt kê các số arstrong nhỏ hơn n. (Mỗi kết quả thỏa mãn cách nhau một khoảng trắng) INPUT

1000

OUTPUT

1 2 3 4 5 6 7 8 9 153 370 371 407

29. 116. BASIC_NUMBERPATTERN_219. Vẽ tam giác số theo nguyên tắc H (1)

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác số theo nguyên tắc tương ứng.

INPUT

5

OUTPUT

~~~1

~~~131

~~13531

~1357531

135797531

30. 117. BASIC_NUMBERPATTERN_220. Vẽ tam giác số theo nguyên tắc H (2)

40

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác số theo nguyên tắc tương ứng.

INPUT

5

OUTPUT

2

242

24642

2468642



31. 118. BASIC_NUMBERPATTERN_221. Vẽ tam giác số theo nguyên tắc H (3)

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác số theo nguyên tắc tương ứng.

INPUT

5

OUTPUT

~~~2

~~~242

~~24642

~2468642

2468108642

32. 119. BASIC_NUMBERPATTERN_222. Vẽ tam giác số theo nguyên tắc I

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác tương ứng.

INPUT

4

OUTPUT

1

23

456

78910

33. 120. BASIC_NUMBERPATTERN_223. Vẽ tam giác số theo nguyên tắc J

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác số theo nguyên tắc tương ứng.

INPUT

5

OUTPUT

1

21

123

4321

12345

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác số theo nguyên tắc tương ứng.
INPUT
5



OUTPUT

```
1
2 6
3 7 10
4 8 11 13
5 9 12 14 15
```

35. 122. BASIC_NUMBERPATTERN_225. Vẽ tam giác số theo nguyên tắc I (2)

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác tương ứng.

(Lưu ý mỗi số ở output cách nhau một khoảng trắng)

INPUT

5

OUTPUT

```
1
2 4
```

7 11 16
22 29 37 46
56 67 79 92 106

36. 123. *BASIC_NUMBERPATTERN_226. Vẽ tam giác số theo nguyên tắc L*

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác tương ứng.

(Lưu ý mỗi số ở output cách nhau một khoảng trắng)

INPUT

4

OUTPUT

1

3 2

42

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

4 5 6
10 9 8 7

37. 124. *BASIC_NUMBERPATTERN_332. Vẽ tam giác ký tự theo nguyên tắc E*
(3)



Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác ký tự theo nguyên tắc tương ứng.

INPUT

4

OUTPUT

ACEG

CEG

EG

G

38. 125. BASIC_NUMBERPATTERN_334. Vẽ tam giác ký tự theo nguyên tắc H

Viết chương trình C cho phép nhập vào chiều cao (số hàng) của tam giác và thực hiện in ra tam giác số theo nguyên tắc tương ứng.

INPUT

5

OUTPUT

@

@B@
@BDB@
@BDFDB@
@BDFHFDB@

4. HÀM VÀ PHẠM VI HOẠT ĐỘNG CỦA BIẾN

4.1. Tính chất của hàm

Hàm (function) hay nói đúng hơn là chương trình con (sub_program) chia cắt các nhiệm vụ tính toán lớn thành các công việc nhỏ hơn và có thể sử dụng nó ở mọi lúc trong chương

trình,
đồng
thời
hàm
cũng
có
thể
được
cung
cấp
cho



nhiều người khác sử dụng dưới dạng thư viện mà không cần phải bắt đầu xây dựng lại từ đầu. Các hàm thích hợp còn có thể che dấu những chi tiết thực hiện đối với các phần khác trong chương trình, vì những phần này không cần biết hàm đó thực hiện như thế nào.

Một chương trình C nói chung bao gồm nhiều hàm nhỏ chứ không phải là một vài hàm lớn. Chương trình có thể nằm trên một hoặc nhiều tệp gốc theo mọi cách thuận tiện; các tệp gốc có thể được dịch tách bạch và nạp vào cùng nhau, cùng với các hàm đã được dịch từ trước trong thư viện. Sau đây là một số tính chất cơ bản của hàm:

- Hàm có thể có kiểu hoặc vô kiểu, kiểu ở đây được hiểu là kiểu giá trị trả về của hàm. Kiểu giá trị trả về của hàm có thể là kiểu cơ bản (base type) hoặc có kiểu do người dùng định nghĩa (user type). Trong trường hợp hàm vô kiểu, C sử dụng từ khoá void để chỉ lớp các hàm kiểu này.

- Hàm có thể có tham số hoặc không có tham số. Trong trường hợp hàm không có tham số C sử dụng từ khoá void để chỉ lớp hàm dạng này. Một lời gọi hàm có nghĩa khi và chỉ khi hàm nhận được đầy đủ giá trị các tham số của nó một cách tường minh.

- Giá trị trả về của hàm được thực hiện bằng lệnh return(giá_trị), giá trị trả về của hàm phải phù hợp với kiểu của hàm. Trong trường hợp hàm vô kiểu, ta có thể sử dụng lệnh return hoặc bỏ qua lệnh return;

- Hàm có thể làm thay đổi nội dung của biến hoặc không làm thay đổi nội dung của biến được truyền cho hàm từ chương trình chính. Nếu ta truyền cho hàm là địa chỉ của biến thì mọi thao tác đối với biến trong hàm đều có thể dẫn tới sự thay đổi nội dung của biến trong chương trình chính, cơ chế này được gọi là cơ chế truyền tham biến cho hàm. Nếu ta truyền cho hàm là nội dung của biến thì mọi sự thay đổi nội dung của biến trong hàm không dẫn tới sự thay đổi nội dung của biến trong chương trình chính, cơ chế này được gọi là cơ chế truyền tham trị.

4.2. Khai báo, thiết kế hàm

Mọi hàm trong C dù là nhỏ nhất cũng phải được thiết kế theo nguyên tắc

sau: Kiểu_hàm Tên_hàm (Kiểu_1 biến_1, Kiểu_2 biến_2, . . .)

{ Khai báo biến cục bộ trong hàm;

Câu_lệnh_hoặc_dãy_câu_lệnh;

return(giá_trị);

}

Ghi chú: Trước khi sử dụng hàm cần phải khai báo nguyên mẫu cho hàm (function prototype) và hàm phải phù hợp với nguyên mẫu của chính nó. Nguyên mẫu của hàm thường được khai báo ở phần đầu chương trình theo cú pháp như sau:

Kiểu_hàm Tên_hàm (Kiểu_1, Kiểu_2 , . . .);



chương trình tìm USCLN của hai số nguyên dương a, b.

```
/* Ví dụ về hàm trả lại một số nguyên int*/  
  
#include <stdio.h>  
  
/* khai báo nguyên mẫu cho hàm; ở đây hàm USCLN trả lại một số nguyên và có hai  
biến kiểu nguyên */  
  
int USCLN( int , int ); /* mô tả hàm */  
  
int USCLN( int a, int b)  
{ while(a!=b){  
    if ( a >b ) a = a -b;  
    else b = b-a; }  
    return(a);}  
  
int main() { /* chương trình chính */  
    unsigned int a, b; clrscr();  
    printf("\n Nhập a ="); scanf("%d", &a);
```

```

printf("\n Nhập b ="); scanf("%d", &b);
printf("\n Ước số chung lớn nhất :
%d",USCLN(a,b)); return 0;
}

```

Ví dụ: Viết hàm chuyển đổi kí tự in hoa thành kí tự in thường.

```

/* Ví dụ về hàm trả lại một kí tự*/
#include <stdio.h>
/* khai báo nguyên mẫu cho hàm; */
char islower(char);
/* mô tả hàm */
char islower ( char c){
    if(c>='A' && c<='Z')

```

```

    c = c + 32;
    return(c);}
/* lời gọi hàm*/
int main(){
    char c='A';

```



```
printf("\n Kí tự được chuyển đổi : %c",  
islower(c)); return 0;}
```

Ví dụ: Viết hàm tính lũy thừa bậc n của số nguyên a.

```
/* Ví dụ về hàm trả lại một số nguyên dài*/  
#include<stdio.h>  
  
/* khai báo nguyên mẫu cho hàm*/  
long power(int , int );  
  
/* mô tả hàm */  
long power ( int a, int n )  
{ long s =1 ; int i;  
  for(i=0; i<n;i++)  
    s*=a;  
  return(s);
```

```

    }

    /* lời gọi hàm */

    int main()

    {
        int a = 5, i;

        for(i=0; i<50;i++)
            printf("\n %d mũ %d = %ld", a , i,
                power(a,i); return 0;
    }

```

Ví dụ 4.20: In ra số nhị phân của một số nguyên.

```

    /* Ví dụ về hàm không trả lại giá trị*/
#include<stdio.h>

    /* khai báo nguyên mẫu cho hàm*/
void binary_int( int );

```

```

    /* mô tả hàm */
void binary_int ( int a)

    {
        int i, k=1; clrscr();
        for(i=15; i>=0; i--)
            { if ( a & (k<<i))

```



```
        printf("%3d", 1);  
    else  
        printf("%3d", 0);  
    }  
}  
  
/* lời gọi hàm */  
int main()  
{  
    int a;  
    printf("\n Nhập a="); scanf("%d", &a);  
    printf("\n Số nhị phân của %d:", a);  
    binary_int(a);  
    return 0;  
}
```

4.3. Phương pháp truyền tham biến cho hàm

Để thấy rõ được hai phương pháp truyền tham trị và truyền tham biến của hàm chúng ta khảo sát ví dụ sau:

Ví dụ: Cho hai số a, b hãy viết hàm đổi chỗ hai số a và b.

```
/* Phương pháp truyền tham trị */
#include<stdio.h>
void swap( float , float );

void swap ( float a, float b)
{
    float temp;
    temp = a;
    a = b;
    b = temp;
}
```

47

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```
int main()
{
    float a = 5, b = 7;
    swap(a, b);
    /* thực hiện đổi chỗ */
}
```



```
printf("\n Giá trị a = %6.2f, b =%6.2f", a, b); }
```

Kết quả thực hiện :

Giá trị của a = 5, b = 7

Nhận xét: Hai biến a, b không được hoán vị cho nhau sau khi thực hiện hàm swap(a,b). Lý do duy nhất để dẫn đến sự kiện này là hàm swap(a,b) thực hiện trên bản sao giá trị của biến a và b. Phương pháp truyền giá trị của biến cho hàm được gọi là phương pháp truyền theo tham trị. Nếu muốn a, b thực sự hoán vị nội dung cho nhau chúng ta phải truyền cho hàm swap(a, b) địa chỉ của ô nhớ của a và địa chỉ ô nhớ của b. Khi đó các thao tác hoán đổi nội dung biến a và b được xử lý trong hàm swap(a, b) thực chất là hoán đổi nội dung của ô nhớ dành cho a thành nội dung ô nhớ dành cho b và ngược lại.

Ví dụ sau sẽ minh họa cơ chế truyền tham biến cho hàm, trước khi chúng ta chưa thảo luận kỹ về con trỏ (pointer), ta tạm ngầm hiểu các qui định như sau:

Toán tử : &(tên_biến) dùng để lấy địa chỉ của biến , chính xác hơn là địa chỉ ô nhớ dành cho biến.

Toán tử : *(tên_biến) dùng để lấy nội dung của ô nhớ dành cho biến.

Ví dụ: Cho hai số a, b hãy viết hàm đổi chỗ hai số a và b.


```

/* Phương pháp truyền tham trị */

#include <stdio.h>
void swap( float , float );
void swap ( float *a, float *b)
{
    float temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    float a = 5, b = 7;

    swap(&a, &b); /* thực hiện đổi chỗ trên địa chỉ của a và
địa chỉ của b*/
    printf("\n Giá trị a = %6.2f b =%6.2f", a, b);
}

```

48

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Kết quả thực hiện :

Giá trị của $a = 7$ $b = 5$

Nhận xét: Giá trị của biến bị thay đổi sau khi hàm swap() thực hiện trên địa chỉ của hai biến a và b. Cơ chế truyền cho hàm theo địa chỉ của biến được gọi là phương pháp truyền tham biến cho hàm. Nếu hàm được truyền theo tham biến thì nội dung của biến sẽ bị thay đổi sau khi thực hiện hàm.

4.4. Biến địa phương, biến toàn cục

a) Biến toàn cục

Biến toàn cục là biến được khai báo ở ngoài tất cả các hàm (kể cả hàm main()). Vùng bộ nhớ



cấp phát cho biến toàn cục được xác định ngay từ khi kết nối (link) và không bị thay đổi trong suốt thời gian chương trình hoạt động. Cơ chế cấp phát bộ nhớ cho biến ngay từ khi kết nối còn được gọi là cơ chế cấp phát tĩnh.

Nội dung của biến toàn cục luôn bị thay đổi theo mỗi thao tác xử lý biến toàn cục trong chương trình con, do vậy khi sử dụng biến toàn cục ta phải quản lý chặt chẽ sự thay đổi nội dung của biến trong chương trình con.

Phạm vi hoạt động của biến toàn cục được tính từ vị trí khai báo nó cho tới cuối văn bản chương trình. Về nguyên tắc, biến toàn cục có thể khai báo ở bất kỳ vị trí nào trong chương trình, nhưng nên khai báo tất cả các biến toàn cục lên đầu chương trình vì nó làm cho chương trình trở nên sáng sủa và dễ đọc, dễ nhìn, dễ quản lý.

Ví dụ: Ví dụ về biến toàn cục

```
/* Ví dụ về biến toàn cục */

#include <stdio.h>

#include <conio.h>

/* khai báo nguyên mẫu cho hàm */

void Tong_int( void );

/* khai báo biến toàn cục */
int a = 5, b=7;

/* mô tả hàm */

int tong(void)

{ printf("\n Nhập a="); scanf("%d",&a);

  printf("\n Nhập b="); scanf("%d",&b);

  return(a+b); }

/* chương trình chính */

int main(){

  printf("\n Giá trị a, b trước khi thực hiện hàm ");

  printf(" a =%5d b = %5d a + b =%5d", a, b, a +

b); printf("\n Giá trị a, b sau khi thực hiện hàm

");

49

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

printf(" a =%5d b = %5d a + b =%5d", a, b, tong());

return 0;

}
```

Kết quả thực hiện:

Giá trị a, b trước khi thực hiện hàm $a = 5$ $b = 7$ $a + b = 12$

Giá trị a, b sau khi thực hiện hàm



Nhập a =
10
Nhập b =
20
a = 10 b
= 20 a +
b = 30

b) Biến địa phương

Biến địa phương là các biến được khai báo trong các hàm và chỉ tồn tại trong thời gian hàm hoạt động.

Tầm tác

dụng của biến địa phương cũng chỉ hạn chế trong hàm mà nó được khai báo, không có mối liên hệ nào giữa biến toàn cục và biến địa phương mặc dù biến địa phương có cùng tên, cùng kiểu với biến toàn cục.

Cơ chế cấp phát không gian nhớ cho các biến địa phương được thực hiện một cách tự động, khi nào khởi động hàm thì các biến địa phương được cấp phát bộ nhớ. Mỗi lần khởi động hàm là một lần cấp phát bộ nhớ, do vậy địa chỉ bộ nhớ dành cho các biến địa phương luôn luôn thay đổi sau mỗi lần gọi tới hàm.

Nội dung của các biến địa phương không được lưu trữ sau khi hàm thực hiện, các biến địa phương sinh ra sau mỗi lần gọi hàm và bị giải phóng ngay sau khi ra khỏi hàm. Các tham số của hàm cũng là biến địa phương. Nghĩa là, tham số của hàm cũng chỉ được khởi động khi gọi tới hàm.

Biến địa phương tĩnh (static): là biến địa phương đặc biệt được khai báo thêm bởi từ khoá static. Khi một biến địa phương được khai báo là static thì biến địa phương được cấp phát một vùng bộ nhớ cố định vì vậy nội dung của biến địa phương sẽ được lưu trữ lại lần sau và tồn tại ngay cả khi hàm đã kết thúc hoạt động. Mặc dù biến toàn cục và biến địa phương tồn tại trong suốt thời gian chương trình hoạt động nhưng điều khác nhau cơ bản giữa chúng là biến toàn cục có thể được truy nhập và sử dụng ở mọi lúc, mọi nơi, còn biến

địa phương static chỉ có tầm hoạt động trong hàm mà nó được khai báo là static.

Ví dụ: Ví dụ về sử dụng biến địa phương static trong hàm

bien_static() chứa biến tĩnh i và kiểm tra nội dung của i sau 5 lần gọi tới hàm. #include<stdio.h>

```
/* nguyên mẫu của hàm */
void bien_static(void);

/* mô tả hàm */
void bien_static(void) {
    static int i; /* khai báo biến static */
```

50

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```
    i++;
    printf("\n Lần gọi thứ %d", i);
}
int main(){
    int n;
    for(n=1; n<=5; n++)
```

```
        bien_static();
    }
```

Kết quả thực hiện:

Lần gọi thứ 1

Lần gọi thứ 2

Lần gọi thứ 3

Lần gọi thứ 4

Lần gọi thứ 5

Biến địa phương dạng thanh ghi (register) : Chúng ta đã biết rằng các bộ vi xử lý đều có các thanh ghi, các thanh ghi nằm ngay trong CPU và không có địa chỉ riêng biệt như các ô nhớ khác trong bộ nhớ chính nên tốc độ xử lý cực nhanh. Do vậy, để



tận dụng ưu điểm về tốc độ của các thanh ghi chúng ta có thể khai báo một biến địa phương có kiểu register. Tuy nhiên, việc làm này cũng nên hạn chế vì số thanh ghi tự do không có nhiều. Nên sử dụng biến thanh ghi trong các trường hợp biến đó là biến đếm trong các vòng lặp.

Ví dụ: Biến địa phương có sử dụng register.

```
#include<stdio.h>

/* nguyên mẫu của hàm */
void bien_static(void);

/* mô tả hàm */
void bien_static(void) {
    static int i;

/* khai báo biến static */
    i++;
    printf("\n Lần gọi thứ %d", i);
}

int main() {
    register int n;

    for(n=1; n<=5; n++)
        bien_static();
}
```

51

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Kết quả thực hiện

Lần gọi thứ 1

Lần gọi thứ 2

Lần gọi thứ 3

Lần gọi thứ 4

Lần gọi thứ 5

4.5.
***Tính
đệ qui
của
hàm***



Một lời gọi hàm được gọi là đệ qui nếu nó gọi đến chính nó. Tính đệ qui của hàm cũng giống như phương pháp định nghĩa đệ qui của qui nạp toán học, hiểu rõ được tính đệ qui của hàm cho phép ta cài đặt rộng rãi lớp các hàm toán học được định nghĩa bằng đệ qui và giảm thiểu quá trình cài đặt chương trình. **Ví dụ:** Nhận xét và cài đặt hàm tính $n!$ của toán học

```
n ! = 1 khi n=0;
(n-1)! * n khi n>=1;

/* chương trình tính n! bằng phương pháp đệ qui
*/ #include<stdio.h>

/* khai báo nguyên mẫu của hàm */
unsigned long GIAI_THUA( unsigned int );

/* mô tả hàm */
unsigned long GIAI_THUA(unsigned int n){
    if (n == 0)
        return(1);
    else
        return ( n * GIAI_THUA(n-1));
}

int main() {
    unsigned int n;
    printf("\ Nh\u00e0p n ="); scanf("%d", &n);
    printf("\n n! = %ld", GIAI_THUA(n));
}
```

Ghi chú: Việc làm đệ qui của hàm cần sử dụng bộ nhớ theo kiểu xếp chồng LIFO (Last In, First Out để chứa các kết quả trung gian, do vậy việc xác định điểm kết thúc quá trình gọi đệ qui là hết sức quan trọng. Nếu không xác định rõ điểm kết thúc của quá trình

chương trình sẽ bị treo vì lỗi tràn stack (stack overflow).

Ví dụ:

Viết đệ qui hàm tìm ước số chung lớn nhất của hai số nguyên dương a, b.

```
int USCLN( int a, int b){
    if( b == 0) return(a);
    else return( USCLN( b, a %b));
}
```

Ví dụ: Giải quyết bài toán kinh điển trong các tài liệu về ngôn ngữ lập trình "bài toán Tháp Hà Nội ". Bài toán được phát biểu như sau:

Có ba cột C1, C2, C3 dùng để xếp đĩa theo thứ tự đường kính giảm dần của các chiếc đĩa. Hãy tìm biện pháp dịch chuyển N chiếc đĩa từ cột C1 sang cột C2 sao cho các điều kiện sau được thoả mãn:

- Mỗi lần chỉ được phép dịch chuyển một đĩa
- Mỗi đĩa có thể được dịch chuyển từ cột này sang một cột khác bất kỳ -

Không được phép để một đĩa trên một đĩa khác có đường kính nhỏ hơn

Ta nhận thấy, với $N = 2$ chúng ta có cách làm như sau: Chuyển đĩa bé nhất (đĩa 1) sang C3, chuyển đĩa còn lại sang C2, chuyển đĩa 1 từ C3 sang C2.

Với $N = 3$ ta lại xử lý lần lượt như sau với giả thiết đã biết cách làm với $N = 2$ ($N - 1$ đĩa): Chuyển đĩa 1, 2 sang C3 theo như cách làm với $N=2$; chuyển đĩa 3 sang cột 2, chuyển đĩa 1 và 2 từ C3 sang C2.

Chúng ta có thể tổng quát hoá phương pháp dịch chuyển bằng hàm sau:

DICH_CHUYEN(N_đĩa, Từ_Cột, Đến_Cột, Cột_Trung_Gian);

Với $N=2$ công việc có thể được diễn tả như sau:

DICH_CHUYEN(1, C1, C3, C2);

DICH_CHUYEN(1, C1, C2, C3);

DICH_CHUYEN(1, C3, C2, C1);

Với $N=3$ công việc dịch chuyển thực hiện như $N=2$ nhưng thực hiện dịch chuyển 2 đĩa

DICH_CHUYEN(2, C1, C3, C2);

DICH_CHUYEN(1, C1, C2, C3);

DICH_CHUYEN(2, C3, C2, C1);

Với N tổng quát ta có :

DICH_CHUYEN($N - 1$, C1, C3, C2);

DICH_CHUYEN(1, C1, C2, C3);

DICH_CHUYEN($N - 1$, C3, C2, C1);

Yêu cầu ban đầu: dịch chuyển N đĩa từ cột C1 sang cột C2 thông qua cột trung gian C3:



C1
C2
C3

Thực hiện:
DICH_CHUYEN(N-1, C1,
C3, C2);

C1 C2 C3

Thực hiện:
DICH_CHUYEN(1, C1,
C2, C3);

C1 C2 C3

Thực hiện:
DICH_CHUYEN(N-1, C3,

C2, C1);

C1 C2 C3

Bài toán Tháp Hà Nội được thể hiện thông qua đoạn chương trình sau:

```
#include <stdio.h>
#include <conio.h>
/* Khai báo nguyên mẫu cho hàm*/
void DICH_CHUYEN (int , int , int , int );
/* Mô tả hàm */
void DICH_CHUYEN (int N, int C1, int C2, int C3)
{ if ( N ==1 )
```



```

printf("\n %5d -> %5d", C1, C2);
else {
    DICH_CHUYEN ( N-1, C1, C3, C2);
    DICH_CHUYEN ( 1, C1, C2, C3);
    DICH_CHUYEN ( N-1, C3, C2, C1);
}
}

```

5.



CẤU TRÚC DỮ LIỆU KIỂU MẢNG (Array)

5.1. *Khái niệm về mảng*

Mảng là một tập cố định các phần tử cùng có chung một kiểu dữ liệu với các thao tác tạo lập mảng, tìm kiếm, truy cập một phần tử của mảng, lưu trữ mảng. Ngoài giá trị, mỗi phần tử của mảng còn được đặc trưng bởi chỉ số của nó thể hiện thứ tự của phần tử đó trong mảng. Không có các thao tác bổ sung thêm vùng nhớ hoặc loại bỏ vùng nhớ của

mảng vì số vùng nhớ cho phần tử trong mảng là cố định.

Một mảng một chiều gồm n phần tử được coi như một vector n thành phần, phần tử thứ i của nó được tương ứng với một chỉ số thứ $i - 1$ đối với ngôn ngữ lập trình C vì phần tử đầu tiên được bắt đầu từ chỉ số 0. Chúng ta có thể mở rộng khái niệm của mảng một chiều thành khái niệm về mảng nhiều chiều.

Một mảng một chiều gồm n phần tử trong đó mỗi phần tử của nó lại là một mảng một chiều gồm m phần tử được gọi là một mảng hai chiều gồm $n \times m$ phần tử.

Tổng quát, một mảng gồm n phần tử mà mỗi phần tử của nó lại là một mảng $k - 1$ chiều thì nó được gọi là mảng k chiều. Số phần tử của mảng k chiều là tích số giữa số các phần tử của mỗi mảng một chiều.

Khai báo mảng một chiều được thực hiện theo qui tắc như sau:

Tên_kiểu Tên_biến[Số_phần_tử];

Ví dụ :

int A[10]; /* khai báo mảng tối đa chứa 10 phần tử nguyên*/

char str[20]; /* khai báo mảng tối đa chứa 19 kí tự */

float B[20]; /* khai báo mảng tối đa chứa 20 số thực */

long int L[20]; /* khai báo mảng tối đa chứa 20 số nguyên dài */

b- Cấu trúc lưu trữ của mảng một chiều

Cấu trúc lưu trữ của mảng: Mảng được tổ chức trong bộ nhớ như một vector, mỗi thành phần của vector được tương ứng với một ô nhớ có kích cỡ đúng bằng kích cỡ của kiểu phần tử và được lưu trữ kế tiếp nhau. Nếu chúng ta có khai báo mảng gồm n phần tử thì phần tử đầu tiên là phần tử thứ 0 và phần tử cuối cùng là phần tử thứ $n - 1$, đồng thời mảng được cấp phát một vùng không gian nhớ liên tục có số byte được tính theo công thức:

Kích_cỡ_mảng = (Số_phần_tử * sizeof(kiểu_phần_tử).

Ví dụ chúng ta có khai báo:

int A[10];

Khi đó kích cỡ tính theo byte của mảng là :

$10 * \text{sizeof}(\text{int}) = 20 \text{ byte};$

float B[20]; => mảng được cấp phát: $20 * \text{sizeof}(\text{float}) = 80 \text{ byte};$



Chương trình dịch của ngôn ngữ C luôn qui định tên của mảng đồng thời là địa chỉ phần tử đầu tiên của mảng trong bộ nhớ. Do vậy, nếu ta có một kiểu dữ liệu nào đó là `Data_type` tên của mảng là `X`, số phần tử của mảng là 10 thì mảng được tổ chức trong bộ nhớ như sau:

`Data_type X[N];`

`X[0] X[1] X[2] X[3] X[N-1]`

`X` - là địa chỉ đầu tiên của mảng.

`X = &X[0] = (X + 0);`

`&X[1] = (X + 1);`

.....

`&X[i] = (X + i);`

Ví dụ: Tìm địa chỉ các phần tử của mảng gồm 10 phần tử nguyên.

`#include<stdio.h>`

```

int main() {
    int A[10], i ;
    /* khai báo mảng gồm 10 biến nguyên */
    printf("\n Địa chỉ đầu của mảng A là : %p", A);
    printf("\n Kích cỡ của mảng : %5d byte", 10 * sizeof(int));
    for ( i =0 ; i <10; i ++){
        printf("\n Địa chỉ phần tử thứ %5d : %p", i, &A[i]);
    }
    return 0;
}

```

Kết quả thực hiện chương trình:

Địa chỉ đầu của mảng: FFE2

Kích cỡ của mảng: 20

Địa chỉ phần tử thứ 0 = FFE2

56

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Địa chỉ phần tử thứ 1 = FFE4

Địa chỉ phần tử thứ 2 = FFE6

Địa chỉ phần tử thứ 3 = FFE8

Địa chỉ phần tử thứ 4 = FFEA

Địa chỉ phần tử thứ 5 = FFEC

Địa chỉ phần tử thứ 6 = FFE2

Địa chỉ
phần tử
thứ 7 =
FFF0

Địa chỉ
phần tử
thứ 8 =
FFF2

Địa chỉ
phần tử
thứ 9 =
FFF4

c- Cầu



trúc lưu trữ mảng nhiều chiều

Ngôn ngữ C không hạn chế số chiều của mảng, chế độ cấp phát bộ nhớ cho mảng nhiều chiều được thực hiện theo cơ chế ưu tiên theo hàng.

Khai báo mảng nhiều chiều :

Data_type tên_biến[số_chiều_1] [số_chiều_2] . . . [số_chiều_n]

Ví dụ:

int A[3][3]; khai báo mảng hai chiều gồm 9 phần tử nguyên được lưu trữ liên tục từ A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2], A[2][0], A[2][1], A[2][2]

Ví dụ: Kiểm tra cấu trúc lưu trữ của bảng hai chiều trong bộ nhớ.

```
#include<stdio.h>

int main() {
    float A[3][3] ;
    /* khai báo mảng hai chiều gồm 9 phần tử
    nguyên*/ int i, j;
    /* Địa chỉ của các hàng*/
    for(i=0; i<3; i++)
        printf("\n Địa chỉ hàng thứ %d là :%p", i,
        A[i]); for(i=0; i<3;i++){
        printf("\n");
        for(j=0;j<3;j++)
            printf("%10p",&A[i][j]);
        }
    return 0;
}
```

Kết quả thực hiện chương trình:

Địa chỉ hàng thứ 0 = FFD2

Địa chỉ hàng thứ 1 = FFDE

Địa chỉ hàng thứ 2 = FFEA

Địa chỉ phần tử A[0][0]= FFD2

Địa chỉ phần tử A[0][1]= FFDE

Địa chỉ
phần tử
A[0][2]=

FFDA

Địa chỉ phần tử A[1][0]= FFDE

Địa chỉ phần tử A[1][1]= FFE2

Địa chỉ phần tử A[1][2]= FFE6

Địa chỉ phần tử A[2][0]= FFEA

Địa chỉ phần tử A[2][1]= FFEE

Địa chỉ phần tử A[2][2]= FFF2

Ví dụ: Kiểm tra cấu trúc lưu trữ của bảng ba chiều trong bộ nhớ.

```
#include<stdio.h>

int main()
{
float A[3][4][5] ;

/* khai báo mảng ba chiều */

    int i, j , k;
    for(i=0; i<3;i++){
        for(j=0;j<4;j++){
            printf("\n");

for( k=0; k <5; k ++)
                printf("%10p",&A[i][j]);
        }

    }

return 0;

}
```

Ghi chú: Kết quả thực hiện ví dụ trên có thể cho ra kết quả khác nhau trên các máy tính khác nhau vì việc phân bổ bộ nhớ cho mảng tùy thuộc vào vùng bộ nhớ tự do của mỗi máy.

5.2. Các thao tác đối với mảng

Các thao tác đối với mảng bao gồm: tạo lập mảng, tìm kiếm phần tử của mảng, lưu trữ

mảng. Các thao tác này có thể được thực hiện ngay từ khi khai báo mảng. Chúng ta có thể vừa khai báo mảng vừa khởi đầu cho mảng, song cần chú ý một số kỹ thuật sau khi khởi đầu cho mảng.

Ví dụ:

```
int A[10] = { 5, 7, 2, 1, 9 };
```

Với cách khai báo và khởi đầu như trên, chương trình vẫn phải cấp phát cho mảng A kích

cỡ $10 * \text{sizeof}(\text{int}) = 20$ byte bộ nhớ, trong khi đó số byte cần thiết thực sự cho mảng chỉ là $5 * \text{sizeof}(\text{int}) = 10$ byte. Để tránh lãng phí bộ nhớ chúng ta có thể vừa khai báo và khởi đầu dưới dạng sau.

```
int A[] = { 5, 7, 2, 1, 9 };
```

Trong ví dụ này, vùng bộ nhớ cấp phát cho mảng chỉ là số các số nguyên được khởi đầu trong dãy $5 * \text{sizeof}(\text{int}) = 10$ byte.

Sau đây là một số ví dụ minh họa cho các thao tác xử lý mảng một và nhiều chiều.

Ví dụ: Tạo lập mảng các số thực gồm n phần tử, tìm phần tử lớn nhất và chỉ số của phần tử lớn nhất trong mảng.

```
#include<stdio.h>

#define MAX 100 /*số phần tử tối đa trong mảng*/

int main() {

    float A[MAX], max; int i, j, n;

    /* Khởi tạo mảng số */

    printf("\n Nhập số phần tử của mảng n="); scanf("%d", &n);
    for(i=0; i<n; i++){
        printf("\n Nhập A[%d] =", i); scanf("%f", &A[i]);
    }

    max = A[0]; j = 0;

    for(i=1; i<n; i++){
        if( A[i]>max) {
            max=A[i]; j = i;
        }
    }

    printf("\n Chỉ số của phần tử lớn nhất là : %d", j);
    printf("\n Giá trị của phần tử lớn nhất là: %6.2f",
max); return 0;

}
```

Kết quả thực hiện chương trình:

Nhập số phần tử của mảng n=7

Nhap A[0]=1

Nhap A[1]=9

Nhap A[2]=2

Nhap A[3]=8

Nhap A[4]=3

Nhap A[5]=7

Nhap

A[6]=4

Chỉ số
của phần
tử lớn
nhất là: 1

Giá trị
của phần
tử lớn
nhất là: 9



Ví dụ: Tạo lập ma trận cấp $m \times n$ và tìm phần tử lớn nhất, nhỏ nhất của ma trận. #include<stdio.h>

```
#define M 20
#define N 20
int main(){
float A[M][N], max, t; int i, j, k, p, m, n;
clrscr();
printf("\n Nhập số hàng của ma trận:"); scanf("%d", &m);
printf("\n Nhập số cột của ma trận:"); scanf("%d", &n);
for(i=0; i<m;i++){
    for(j=0; j<n ; j++){
```



```

        printf("\n Nhập A[%d][%d] =", i,j);
        scanf("%f", &t); A[i][j]=t;
    }
}
max=A[0][0]; k=0; p=0;
for(i=0; i<m; i++){
    for(j=0;j<n; j++){
        if(A[i][j]>max) {
            max=A[i][j]; k=i ; p =j;
        }
    }
}
printf("\n Phần tử có giá trị max là A[%d][%d] = %
6.2f", k,p, max);
return 0;

```

60

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

}

Ghi chú: C không hỗ trợ khuôn dạng nhập dữ liệu %f cho các mảng nhiều chiều. Do vậy, muốn nhập dữ liệu là số thực cho mảng nhiều chiều chúng ta phải nhập vào biến trung gian sau đó gán giá trị trở lại. Đây không phải là hạn chế của C++ mà hàm scanf() đã được thay thế bởi toán tử "cin". Tuy nhiên, khi sử dụng cin, cout chúng ta phải viết chương trình dưới dạng *.cpp.

5.3. Mảng và đối của hàm

Như



chúng ta đã biết, khi hàm được truyền theo tham biến thì giá trị của biến có thể bị thay đổi sau mỗi lời gọi hàm. Hàm được gọi là truyền theo tham biến khi chúng ta truyền cho hàm là địa chỉ của biến. Ngôn ngữ C qui định, tên của mảng đồng thời là địa chỉ của mảng trong bộ nhớ, do vậy nếu chúng ta truyền cho hàm là tên của một mảng thì hàm luôn thực hiện theo cơ chế truyền theo tham biến, trường hợp này giống như ta sử dụng từ khoá var trong khai báo biến của hàm trong Pascal. Trong trường hợp muốn truyền theo tham trị với đối số của hàm là một mảng, thì ta phải thực hiện trên một bản sao khác của mảng khi đó các thao tác đối với mảng thực chất đã được thực hiện trên một vùng nhớ khác dành cho bản sao của mảng.

Ví dụ: Tạo lập và sắp xếp dãy các số thực A1, A2, ... An theo thứ tự tăng dần.

Để giải quyết bài toán, chúng ta thực hiện xây dựng chương trình thành 3 hàm riêng biệt, hàm Init_Array() có nhiệm vụ tạo lập mảng số A[n], hàm Sort_Array() thực hiện việc sắp xếp dãy các số được lưu trữ trong mảng, hàm In_Array() in lại kết quả sau khi mảng đã được sắp xếp.

```
#include<stdio.h>

#define MAX 100

/* Khai báo nguyên mẫu cho hàm */
void Init_Array ( float A[], int n);
void Sort_Array( float A[], int n);
void In_Array( float A[], int n);

/* Mô tả hàm */
/* Hàm tạo lập mảng số */
void Init_Array( float A[], int n) {
    int i;
    for( i = 0; i < n; i++ ) {
        printf("\n Nhập A[%d] = ", i);
        scanf("%f", &A[i]);
    }
}
```

```
/* Hàm sắp xếp mảng số */
void Sort_Array( float A[], int n ){
    int i , j ;
    float temp;
    for(i=0; i<n - 1 ; i ++ ) {
        for( j = i + 1; j < n ; j ++ ){
```



```
if ( A[i] >A[j]) {  
    temp = A[i];  
    A[i] = A[j];  
    A[j] = temp;  
}  
}  
}  
  
}  
  
/* Hàm in mảng số  
*/  
void In_Array (  
float A[], int n) {  
  
int i;  
for(i=0; i<n; i++)
```

```
printf("\n Phần tử A[%d] = %6.2f", i, A[i]);  
return 0;  
  
}  
/* Chương trình chính */  
int main() {  
  
    float A[MAX]; int n;  
    printf("\n Nhập số phần tử của mảng n = ");    scanf("%d",  
        &n);  
    Init_Array(A, n);  
    Sort_Array(A,n);  
    In_Array(A, n);  
  
}
```

Ví dụ: Viết chương trình tính tổng của hai ma trận cùng cấp.

Chương trình được xây dựng thành 3 hàm, hàm Init_Matrix() : Tạo lập ma trận cấp m x n; hàm Sum_Matrix() tính tổng hai ma trận cùng cấp; hàm Print_Matrix() in ma trận kết quả. Tham biến được truyền vào cho hàm là tên ma trận, số hàng, số cột của ma trận.

```
#include <stdio.h>
#include <dos.h> /* khai báo sử dụng hàm delay() trong chương
trình*/
#define M 20 /* Số hàng tối đa của ma trận*/
#define N 20 /* Số cột tối đa của ma trận */ /*
Khai báo nguyên mẫu cho hàm*/
```

62

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```
void Init_Matrix(float A[M][N], int m, int n, char ten); void
Sum_Matrix(float A[M][N], float B[M][N], float C[M][N], int
m, int n);
void Print_Matrix(float A[M][N], int m, int
n); /*Mô tả hàm */
void Init_Matrix(float A[M][N], int m, int n, char ten) {
    int i, j; float temp; clrscr();
```

```
for(i=0;
i<m;
i++){
for(j=0;
j<n;
j++){
```



```

        printf("\n Nhập %c[%d][%d] =", ten, i,j);
        scanf("%f", &temp); A[i][j]=temp;
    }
}

}

void Sum_Matrix(float A[M][N],float B[M][N], float C[M][N],
int m,int n){
    int i, j;
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            C[i][j]=A[i][j] + B[i][j];
        }
    }
}

void Print_Matrix(float A[M][N], int m, int n) {
    int i, j , ch=179; /* 179 là mã kí tự '|' */
    for(i=0; i<m; i++){
        printf("\n %-3c", ch);
        for(j=0; j<n; j++){
            printf(" %6.2f", A[i][j]);
        }
        printf("%3c", ch);
    }
    return 0;
}

/* Chương trình chính */
int main() {
    float A[M][N], B[M][N], C[M][N];
    int n, m; clrscr();
    printf("\n Nhập số hàng m ="); scanf("%d", &m);

```

```

    printf("\n Nhập số cột n ="); scanf("%d", &n);
    Init_Matrix(A, m, n, 'A');
    Init_Matrix(B, m, n, 'B');
    Sum_Matrix(A, B, C, m, n);
    Print_Matrix(C, m, n);
}

```

kí tự (string)

Xâu kí tự là một mảng trong đó mỗi phần tử của nó là một kí tự, kí tự cuối cùng của xâu được dùng làm kí tự kết thúc xâu. Kí tự kết thúc xâu được ngôn ngữ C qui định là kí tự '\0', kí tự này có mã là 0 (NULL) trong bảng mã ASCII. Ví dụ trong khai báo :

```
char str[]='ABCDEF'
```

Khi đó xâu kí tự được tổ chức như sau:

0 1 2 3 4 5 6

A B C D E F '\0'

Khi đó `str[0] = 'A'; str[1] = 'B'; . . . , str[5]='F', str[6]='\0';`

Vì kí hiệu kết thúc xâu có mã là 0 nên chúng ta có thể kiểm chứng tổ chức lưu trữ của xâu thông qua đoạn chương trình sau:

Ví dụ:

```
/* In ra từng kí tự trong xâu */
#include <stdio.h>
#include <conio.h>
#include <string.h>

/* sử dụng hàm xử lý xâu kí tự gets() */

int main() {
    char str[20]; int i = 0; char c;
    printf("\n Nhập xâu kí tự:"); gets(str); /* nhập xâu kí tự từ bàn phím */
    while ( str[i]!='\0'){
        putchar(c); i++;
    }
}
```

Ghi chú: Hàm `getch()` nhận một kí tự từ bàn phím, hàm `putch(c)` đưa ra màn hình một kí tự `c`. Hàm `sacnf("%s", str)` : nhận một xâu kí tự từ bàn phím nhưng không được chứa kí tự trống (space), hàm `gets(str)` : cho phép nhận từ bàn phím một xâu kí tự kể cả dấu trống. Ngôn ngữ C không cung cấp các phép toán trên xâu kí tự, mà mọi thao tác trên xâu kí tự đều phải được thực hiện thông qua các lời gọi hàm. Sau đây là một số hàm xử lý xâu kí tự

thông dụng được khai báo trong tệp `string.h`:

puts (string) : Đưa ra màn hình một string.

gets(string) : Nhận từ bàn phím một string.

scanf("%s", string): Nhận từ bàn phím một string không kể kí tự trống (space) .

strlen(string): Hàm trả lại một số là độ dài của string.

strcpy(s,p) : Hàm copy chuỗi p vào chuỗi s.

strcat(s,p) : Hàm nối chuỗi p vào sau chuỗi s.



strcmp(s,p): Hàm trả lại giá trị dương nếu chuỗi s lớn hơn chuỗi p, trả lại giá trị âm nếu chuỗi s nhỏ hơn chuỗi p, trả lại giá trị 0 nếu chuỗi s đúng bằng chuỗi p.

strstr(s,p) : Hàm trả lại vị trí của chuỗi p trong chuỗi s, nếu p không có mặt trong s hàm trả lại con trỏ NULL.

strncmp(s,p,n) : Hàm so sánh n ký tự đầu tiên của chuỗi s và p.

strncpy(s,p,n) : Hàm copy n ký tự đầu tiên từ chuỗi p vào chuỗi s.

strrev(str): Hàm đảo chuỗi s theo thứ tự ngược lại.

Chúng ta có thể sử dụng trực tiếp các hàm xử lý chuỗi ký tự bằng việc khai báo chỉ thị `#include <string.h>`, tuy nhiên chúng ta có thể viết lại các thao tác đó thông qua ví dụ sau:

Ví dụ: Xây dựng các thao tác sau cho string:

F1- Nhập chuỗi ký tự từ bàn phím hàm `gets(str)`.

F2- Tìm độ dài chuỗi ký tự `strlen(str)`.

F3- Tìm vị trí ký tự C đầu tiên xuất hiện trong chuỗi ký tự.

F4- Đảo xâu kí tự.

F5- Đổi xâu kí tự từ in thường thành in hoa.

F6- Sắp xếp xâu kí tự theo thứ tự tăng dần. . .

```
/* Các thao tác với xâu kí tự */
#include<stdio.h>
#include<dos.h>
#define F1 59
#define F2 60
#define F3 61
#define F4 62
#define F5 63
#define F6 64
#define F7 65
#define F10 68
#define MAX 256
/* khai báo nguyên mẫu cho hàm */
char *gets (char str[]); /* char * được hiểu là một xâu kí
tự */
```

65

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```
int strlen(char str[]); /* hàm trả lại độ dài xâu */ int
strcstr(char str[], char c); /* hàm trả lại vị trí kí tự c
đầu tiên trong str*/
char *strrev(char str[]);/* hàm đảo xâu str*/
char *upper(char str[]); /* hàm đổi xâu str thành chữ in
hoa*/
char *sort_str(char str[]); /* hàm sắp xếp xâu theo thứ tự
tử
```




```

điễn*/
void thuc_hien(void);
/* Mô tả hàm */
/* Hàm trả lại một xâu kí tự được nhập từ bàn
phím*/ char *gets( char str[] ) {
    int i=0; char c;
    while ( ( c=getch())!='\n') { /* nhập nếu không phải
phím enter*/
        str[i] = c; i++;
    }
    str[i]='\0';
    return(str);
}
/* Hàm tính độ dài xâu kí tự: */
int strlen(char str[]) {
    int i=0;
    while(str[i]) i++;
    return(i);
}
/* Hàm trả lại vị trí đầu tiên kí tự c trong xâu
str*/ int strchr (char str[] , char c) {
    int i =0;
    while (str[i] && str[i] != c )
        i++;
    if(str[i]=='\0' ) return(-1);
    return(i);
}
/* Hàm đảo xâu kí tự */
char *strrev( char str[]) {
    int i , j , n=strlen(str); char c;
    i = 0; j = n-1;
    while (i < j) {
        c = str[i] ; str[i] = str [j] ; str[j] =c;
    }
    return(str);
}

```

```

/* Hàm đổi xâu in thường thành in hoa */
char * upper( char str[] ) {
    int i, n=strlen(str);
    for(i=0;i<n; i++){
        if( str[i]>='a' && str[i]<='z')
            str[i]=str[i] - 32;
    }
}

```

```
}  
return(str);
```

```
}  
/*  
Hàm  
sắp  
xếp  
xâu  
kí  
tự  
*/  
char
```



```
*sort_str( char str[] ) {  
    int i, j , n = strlen(str); char temp;  
    for (i =0; i<n-1; i++){  
        for(j=i+1; j<n; j ++){  
            if(str[i] >str[j]){  
                temp = str[i]; str[i] = str[j];  
                str[j] = temp;  
            }  
        }  
    }  
}  
  
/* Hàm thực hiện chức năng */  
void thuc_hien(void) {  
    char c , phim , str[MAX]; int control = 0;  
    textmode(0) ;  
    do {
```

```

clrscr();
printf("\n Tập thao tác với string");
printf("\n F1- Tạo lập string");
printf("\n F2- Tính độ dài xâu");
printf("\n F3- Tìm kí tự trong string");
printf("\n F4- Đảo ngược string");
printf("\n F5- Đổi thành in hoa");
printf("\n F6- Sắp xếp string");
printf("\n F10- Trở về");
phim = return 0;
switch(phim) {
    case F1: gets(str); control=1; break;
    case F2:
        if (control)
            printf("\n Độ dài xâu là:%d",
strlen(str));

```

67

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```

        break;
    case F3:
        if (control){
            printf("\n Kí tự cần tìm:");
            scanf("%c", &c);
            if(strcstr(str, c)>=0)
                printf("\n Vị
trí:%d", strcstr(str,c));
        }
    break;
    case F4:
        if (control)
            printf("\n Xâu đảo:%s",
strrev(str));
        break;
    case F5:
        if (control)
            printf("\n In hoa:%s",
                upper(str));
        break;
    case F6:
        if (control)
            printf("\n Xâu được sắp
                xếp:%s",
sort_str(str));
        break;

```



```

    }
    delay(2000);
} while(phim!=F10);
}
/* chương trình chính */
int main() {
    thuc_hien();
}

```

Mảng các chuỗi: mảng các chuỗi là một mảng mà mỗi phần tử của nó là một chuỗi.

Ví dụ char buffer[25][80] sẽ khai báo mảng các chuỗi gồm 25 hàng trong đó mỗi hàng gồm 80 ký tự. Ví dụ sau đây sẽ minh họa cho các thao tác trên mảng các string.

Ví dụ: Hãy tạo lập mảng các chuỗi trong đó mỗi chuỗi là một từ khóa của ngôn ngữ lập trình C. Sắp xếp mảng các từ khóa theo thứ tự từ điển.

Chương trình sau sẽ được thiết kế thành 3 hàm chính: Hàm Init_KeyWord() thiết lập bảng từ khóa, hàm Sort_KeyWord() sắp xếp mảng từ khóa, hàm In_KeyWord() in mảng các từ khóa.

68

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Chương trình được thực hiện như sau:

```

/* Thao tác với mảng các string */
#include<stdio.h>

#include<dos.h>
/* Khai báo nguyên mẫu cho hàm*/
void Init_KeyWord( char key_word[][20], int

```



```

n); void Sort_KeyWord(char key_word[][20], int
n); void In_KeyWord(char key_word[][20], int
n);
/* Mô tả hàm */
void Init_KeyWord( char key_word[][20], int n) {
    int i;
    for( i = 0; i < n; i++){
        printf("\n Nhập từ khoá %d :",i);
        scanf("%s", key_word[i]);
    }
}
void Sort_KeyWord(char key_word[][20], int n)
{ int i, j; char temp[20];
  for( i = 0; i < n - 1; i++){
      for( j = i +1; j < n; j++){
          if ( strcmp(key_word[i], key_word[j] ) > 0 ){
              strcpy(temp, key_word[i] );
              strcpy(key_word[i], key_word[j] );
              strcpy(key_word[j], temp );
          }
      }
  }
}
void In_KeyWord(char key_word[][20], int n) {
    int i;
    for ( i = 0; i < n; i++){
        printf("\n Key_Word[%d] = %s", i,
        key_word[i]); }
    return 0;
}
int main() {
    char key_word[100][20]; int n;
    printf("\n Nhập số từ khoá n = "); scanf("%d",
    &n); Init_KeyWord(key_word, n);
    Sort_KeyWord(key_word, n);
    In_KeyWord(key_word, n); }

```

5.5. Kiểu dữ liệu Con trỏ

Con trỏ là biến chứa địa chỉ của một biến khác. Con trỏ được sử dụng rất nhiều trong C, sự mềm dẻo của con trỏ trở thành một thế mạnh để biểu diễn tính toán và

truy nhập gián tiếp các đối tượng. Con trỏ đã từng bị coi có hại chẳng kém gì câu lệnh goto vì khi viết chương trình có sử dụng con trỏ sẽ tạo nên các chương trình



tương đối khó hiểu. Tuy nhiên, nếu chúng ta có biện pháp quản lý tốt thì dùng con trỏ sẽ làm cho chương trình trở nên đơn giản và góp phần cải thiện đáng kể tốc độ chương trình.

5.5.1 Con trỏ và địa chỉ

Khai báo con trỏ :

Kiểu_dữ_liệu * Biến_con_trỏ;

Vì con trỏ chứa địa chỉ của đối tượng nên có thể thâm nhập vào đối tượng “gián tiếp” qua con trỏ. Giả sử x là một biến, chẳng hạn là biến kiểu int, giả sử px là con trỏ, được tạo ra theo một cách nào đó. Phép toán một ngôi & cho địa chỉ của đối tượng cho nên câu lệnh

px = &x;

sẽ gán địa chỉ của x cho biến px; px bây giờ được gọi là “trỏ tới” x. Phép toán & chỉ áp dụng được cho các biến và phần tử mảng; kết cấu kiểu &(x + 1) và &3 là không hợp lệ.

Phép toán một ngôi * coi toán hạng của nó là địa chỉ cần xét và thâm nhập tới

địa chỉ đó để lấy ra nội dung của biến. Ví dụ, nếu y là int thì

```
y = *px;
```

sẽ gán cho y nội dung của biến mà px trỏ tới. Vậy đây

```
px = &x;
```

```
y = *px;
```

sẽ gán giá trị của x cho y như trong lệnh

```
y = x;
```

Cũng cần phải khai báo cho các biến tham dự vào việc này:

```
int x, y;
```

```
int *px;
```

Khai báo của x và y là điều ta đã biết. Khai báo của con trỏ px có điểm mới

```
int *px;
```

có ngụ ý rằng đó là một cách tượng trưng; nó nói lên rằng tổ hợp *px có kiểu int, tức là nếu px xuất hiện trong ngữ cảnh *px thì nó cũng tương đương với biến có kiểu int.

70

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Con trỏ có thể xuất hiện trong các biểu thức. Chẳng hạn, nếu px trỏ tới số nguyên x thì *px có thể xuất hiện trong bất kì ngữ cảnh nào mà x có thể xuất hiện

```
y = *px + 1;
```

sẽ đặt y lớn hơn x 1 đơn vị;

```
printf("%d \n", *px);
```

in ra giá trị hiện tại của x.

phép toán một ngôi * và & có mức ưu tiên cao hơn các phép toán số học, cho nên biểu thức này lấy bất kì giá trị nào mà px trỏ tới, cộng với 1 rồi gán cho y.



Con trỏ cũng có thể xuất hiện bên vế trái của phép gán. Nếu px trỏ

tới x thì ***px = 0;** sẽ đặt x thành không và ***px += 1;** sẽ tăng x lên như trong trường hợp **(*px) ++;**

Các dấu ngoặc là cần thiết trong ví dụ cuối; nếu không có chúng thì biểu thức sẽ tăng px thay cho việc tăng ở chỗ nó trở tới, bởi vì phép toán một ngôi như * và + được tính từ phải sang trái.

Cuối cùng vì con trỏ là biến nên ta có thể thao tác chúng như đối với các biến khác. Nếu py là con trỏ nữa kiểu int, thì:

py = px; sẽ sao nội dung của px vào py, nghĩa là làm cho py trở tới nơi mà px trỏ. Ví dụ sau minh họa những thao tác truy nhập gián tiếp tới biến thông qua con trỏ.

Ví dụ 3.10: Ví dụ sau sẽ thay đổi nội dung của hai biến a và b thông qua con trỏ.

```
#include <stdio.h>
#include <conio.h>
int main(){
    int a = 5, b = 7; /* giả sử có hai biến nguyên a =5, b = 7*/
    int *px, *py; /* khai báo hai con trỏ kiểu int */
    px = &a; /* px trỏ tới x */
    printf("\n Nội dung con trỏ px =%d", *px);
    *px = *px + 10; /* Nội dung của *px là 15*/
    /* con trỏ px đã thay đổi nội dung của a */
    printf("\n Giá trị của a = %d", a);
    px = &b; /* px trỏ tới b */
    py = px;
    /* con trỏ py thay đổi giá trị của b thông qua con trỏ px*/
    *py = *py + 10;
    printf("\n Giá trị của b=%d", b);
}
```

Kết quả thực hiện chương trình:

71

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

Nội dung con trỏ px : 5

Giá trị của a : 15

Giá trị của b : 17

5.5.2 Con trỏ và đối của hàm

Ta sẽ phải làm thế nào nếu phải thay đổi nội dung của đối; Chẳng hạn, chương

trình sắp xếp rất có thể dẫn tới việc đổi chỗ hai phần tử chưa theo thứ tự thông qua một hàm là



swap. Viết như sau thì chưa đủ

```
swap(a, b);
```

với hàm swap được định nghĩa như sau: swap(x, y) /*sai*/

```
void swap(int x, int y) {  
    int temp; temp = x; x = y; y = temp;  
}
```

Do việc hàm gọi theo giá trị nên swap không làm ảnh hưởng tới các đối a và b trong hàm gọi tới nó, nghĩa là không thực hiện được việc đổi chỗ. Để thu được kết quả mong muốn, chương trình gọi sẽ truyền con trỏ tới giá trị cần phải thay đổi

```
swap(&a, &b);
```

Vì phép toán & cho địa chỉ của biến, nên &a là con trỏ tới a. Trong bản thân swap, các đối được khai báo là con trỏ, và toán hạng thực tại sẽ thâm nhập qua chúng.

```
void swap(int *px, int *py) {  
    int temp;
```

```
temp = *px; *px = *py; *py = temp;
}
```

Con trỏ thường được sử dụng trong các hàm phải cho nhiều hơn một giá trị (có thể nói rằng swap cho hai giá trị, các giá trị mới thông qua đối của nó).

5.5.3 Con trỏ và mảng

Mảng trong C thực chất là một hằng con trỏ, do vậy, mọi thao tác đối với mảng đều có thể được thực hiện thông qua con trỏ.

Khai báo

```
int a[10];
```

xác định mảng có kích thước 10 phần tử int, tức là một khối 10 đối tượng liên tiếp có tên a[0], a[1],...a[9]. Cú pháp a[i] có nghĩa là phần tử của mảng ở vị trí thứ i kể từ vị trí đầu. Nếu pa là con trỏ tới một số nguyên, được khai báo là

```
int *pa;
```

thì phép gán

```
pa = &a[0];
```

72

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

sẽ đặt pa để trỏ tới phần tử đầu của a; tức là pa chứa địa chỉ của a[0]. Bây giờ phép gán

```
x = *pa;
```

sẽ sao nội dung của a[0] vào x.

Nếu pa trỏ tới một phần tử của bảng a thì theo định nghĩa, pa + 1 sẽ trỏ tới phần tử tiếp theo và pa - i sẽ trỏ tới phần tử thứ i trước pa, pa + i sẽ trỏ tới phần tử thứ i sau pa. Vậy nếu pa trỏ tới a[0] thì

***(pa + 1)**

sẽ cho nội dung của a[1], pa+i là địa chỉ của a[i] còn *(pa + i) là nội dung của a[i].

Các lưu ý này là đúng bất kể đến kiểu của biến trong mảng a. Định nghĩa “cộng 1 vào con trỏ” sẽ được lấy theo



tỉ lệ kích thước lưu trữ của đối tượng mà pa trở tới. Vậy trong $pa + i$, i sẽ được nhân với kích thước của đối tượng mà pa trở tới trước khi được cộng vào pa.

Sự tương ứng giữa việc định chỉ số và phép toán số học trên con trỏ là rất chặt chẽ. Trong thực tế, việc tham trỏ tới mảng được trình biên dịch chuyển thành con trỏ tới điểm đầu của mảng. Kết quả là tên mảng chính là một biểu thức con trỏ. Vì tên mảng là đồng nghĩa với việc định vị phần tử thứ 0 của mảng a nên phép gán

pa = &a[0];

cũng có thể viết là

pa = a;

Điều cần chú ý là để tham trỏ tới $a[i]$ có thể được viết dưới dạng $*(a + i)$. Trong khi tính $a[i]$, C chuyển tức khắc nó thành $*(a + i)$; hai dạng này hoàn toàn là tương đương nhau. Áp dụng phép toán & cho cả hai dạng này ta có $\&a[i]$ và $(a + i)$ là địa chỉ của phần tử thứ i trong mảng a. Mặt khác nếu pa là con trỏ thì các biểu thức có thể sử dụng nó kèm thêm chỉ số: $pa[i]$ đồng nhất với $*(pa + i)$. Tóm lại, bất kì một mảng và biểu thức chỉ số nào cũng đều được viết như một con trỏ.

Có một sự khác biệt giữa tên mảng và con trỏ cần phải nhớ. Con trỏ là một biến, nên $pa = a$ và $pa ++$ đều là các phép toán đúng, còn tên mảng là một hằng chứ không là biến: kết cấu kiểu $a = pa$ hoặc $a ++$ hoặc $p = \&a$ là không hợp lệ.

Khi truyền một tên mảng cho hàm thì tên của mảng là địa chỉ đầu của mảng, do vậy, tên mảng thực sự là con trỏ. Ta có thể dùng sự kiện này để viết ra bản mới của `strlen`, tính chiều dài của chuỗi ký tự.

```
int strlen( char * s) /*cho chiều dài của chuỗi s*/  
{  
  
    int n;  
  
    for (n = 0; *s != '\0'; s ++)  
        n ++;  
  
    return(n);  
  
}
```

73

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

}

Việc tăng s là hoàn toàn hợp pháp vì nó là biến con trỏ; $s ++$ không ảnh hưởng tới chuỗi ký tự trong hàm gọi tới `strlen` mà chỉ làm tăng bản sao riêng của địa chỉ trong `strlen`.

Vì các tham biến hình thức trong định nghĩa hàm

char s[]; và **char *s;** là hoàn toàn tương đương. Khi truyền một tên mảng cho hàm, hàm có thể

coi rằng nó xử lý hoặc mảng hoặc con trỏ là giống nhau. Một khía cạnh khác, nếu p và q trỏ tới các thành phần của cùng một mảng thì có thể áp dụng được các quan hệ như <, <=, >= v.v... chẳng hạn

p < q

là đúng, nếu p con trỏ tới thành phần đứng trước thành phần mà q trỏ tới trong mảng. Các quan hệ == và != cũng áp dụng được. Có thể so sánh một con trỏ với giá trị NULL. Nhưng so sánh các con trỏ trỏ tới các mảng khác nhau sẽ không đưa lại kết quả mong muốn.

Tiếp nữa, chúng ta đã quan sát rằng con trỏ và số nguyên có thể cộng hoặc trừ cho nhau. Kết cấu

p + n

nghĩa là đối tượng thứ n sau đối tượng do p trỏ tới. Điều này đúng với bất kể loại đối tượng nào mà p được khai báo sẽ trỏ tới; trình biên dịch sẽ tính tỉ lệ n theo kích thước của các đối tượng do p trỏ tới, điều này được xác định theo khai báo của p. Chẳng hạn trên PC AT, nhân từ tỉ lệ là 1 đối với char, 2 đối với int và short, 4 cho long và float.

Phép trừ con trỏ cũng hợp lệ; nếu p và q trỏ tới các thành phần của cùng một mảng thì p - q là số phần tử giữa p và q. Dùng sự kiện này ta có thể viết ra một bản khác cho strlen:

```
/*cho độ dài chuỗi s*/
```

```
int strlen( char *s) {
```

```
    char *p = s;
```

```
    while (*p != '\0')
```

```
        p ++;
```

```
    return(p-s);
```

```
}
```

Trong khai báo, p được khởi đầu là s, tức là trỏ tới ký tự đầu tiên. Chu trình while, kiểm tra lần lượt từng ký tự xem đã là '\0' chưa để xác định cuối chuỗi. Vì '\0' là 0 và vì while chỉ kiểm tra xem biểu thức có là 0 hay không nên có thể bỏ phép thử tường minh, vậy ta có thể viết lại chu trình trên

```
    while (*p)
```

```
        p ++;
```

Do p trỏ tới các ký tự nên p ++ sẽ chuyển p để trỏ sang ký tự tiếp theo, và p - s sẽ cho số các ký tự đã lướt qua, tức là độ dài của chuỗi. Ví dụ sau minh họa rõ ràng về phương pháp sử dụng con trỏ.

Ví dụ 3.11- Viết chương trình sắp xếp dãy các số nguyên theo thứ tự tăng dần

bằng con trỏ.

Chương trình được thiết kế thành 3 hàm chính, hàm `Init_Array()` tạo lập dãy các số thực, hàm `Sort_Array()` sắp xếp dãy các số thực theo thứ tự tăng dần, hàm



`In_Array()` in kết quả đã được sắp xếp.

```
#include <stdio.h>
#include <conio.h>
#define MAX 100
/* Khai báo nguyên mẫu cho hàm */
void Init_Array( float *A, int n);
void Sort_Array( float *A, int n);
void In_Array( float *A, int n);
/* Mô tả hàm*/
void Init_Array( float *A, int n) {
    int i;
    for(i=0; i<n;i++){
```

```

        printf("\n Nhập A[%d] = ", i);
        scanf("%f", (A + i) );
    }
}

void Sort_Array( float *A, int n) {
    int i, j; float temp;
    for ( i =0; i< n -1; i++){
        for( j = i +1; j <n; j++){
            if( *(A+i) > *(A + j) ) {
                temp = *( A +i );
                *(A + i ) = * (A + j );
                *(A +j) = temp;
            }
        }
    }
}

```

```

    }
}

void In_Array( float *A, int n) {
    int i;
    for(i=0; i<n;i++){

```



```
        printf("\n Phần tử A[%d] = % 6.2f", i, *(A +i) );  
    }  
}  
/* chương trình chính */  
int main() {  
    float A[MAX]; int n;  
    printf("\n Nhập n="); scanf("%d", &n);  
    Init_Array(A,n);  
    Sort_Array(A,n);  
    In_Array(A,n);  
}
```

5.5.4 Cấp phát bộ nhớ cho con trỏ

Nếu p là con trỏ thì p ++ sẽ tăng p để trỏ tới phần tiếp của đối tượng mà p đã

trở tới , bất kể kiểu đối tượng là gì, còn $p++$ sẽ tăng lên trở tới phần tử thứ i kể từ sau phần tử p trở tới. Đa số các trường hợp chúng ta đang xét thì p được trở tới một mảng nào đó đã được khai báo trước, khi đó mặc nhiên vùng bộ nhớ mà con trỏ xử dụng là vùng bộ nhớ đã được cấp phát cho mảng trước đó và ta có thể xử lý như một mảng. Tuy nhiên, chúng ta có thể sử dụng con trỏ linh động hơn bằng các phương pháp cấp phát bộ nhớ động cho con trỏ.

Trong thư viện chuẩn của C cung cấp cho ta một số hàm cấp phát bộ nhớ cho con trỏ

Hàm **malloc(size_t size)** : cấp phát một vùng bộ nhớ có kích cỡ size cho con trỏ. Hàm trả lại con trỏ NULL nếu size = 0;

Hàm **calloc(n, i)** : đặt con trỏ vào đầu vùng bộ nhớ gồm $n \times i$ bytes tự do. Hàm trả lại con trỏ NULL nếu việc cấp phát không thành công.

Hàm farmalloc(), farcalloc() : đặt con trỏ vào đầu vùng bộ nhớ tự do ngoài vùng heap, trả lại con trỏ NULL nếu việc cấp phát không thành công.

Hàm free(p) : giải phóng vùng bộ nhớ đã cấp phát bởi malloc() hoặc calloc() trước đó.

Hàm farfree() : giải phóng vùng bộ nhớ đã cấp phát bởi farmalloc() hoặc farcalloc() trước đó.

Ví dụ 3.12: Xây dựng các thao tác tạo lập đa thức, tính giá trị đa thức theo lược đồ Hooneir, tính tổng hai đa thức, hiệu hai đa thức.

```
/* Chương trình xây dựng tập thao tác cho đa thức */  
#include <stdio.h>
```




```
#include <dos.h>
#include <alloc.h>
#define F1 59
#define F2 60
#define F3 61
#define F4 62
#define F5 63
#define F10 68
/* Nguyên mẫu của hàm */
void Init_Dathuc(float *A, int n, char c);
void In_Dathuc(float *A, int n);
float Val_Dathuc( float *A, int n);
void Tong_Dathuc(float *C, float *A, float *B, int n,
int m);
```

```

void Hieu_Dathuc(float *C,float *A, float *B, int n, int
m);
void Thuc_Hien(void);
/* Mô tả hàm */
void Init_Dathuc(float *A, int n, char c){
    int i;float t;
    for(i=0;i<n;i++){
        printf("\n %c[%d]=",c,i);
        scanf("%f",&t);
        A[i]=t;
    }
    In_Dathuc(A,n);
}
void In_Dathuc(float *A, int n){
    int i;

```

77

Phan Thị Hà-Khoa CNTT1-Học viện CNBCVT

```

        for(i=0;i<n;i++)
            printf("%7.2f",A[i]);
        printf("\n");
        delay(2000);
    }
float Val_Dathuc( float *A, int n){

```

float



```

    s=A[n-1], k; int i, j;
    printf("\n Nhap k="); scanf("%f",&k);
    for(i=n-2; i>=0; i--)
        s = s*k + A[i];
    printf("\n S=%6.2f", s); delay(2000);
    return(s);
}

void Tong_Dathuc(float *C, float *A, float *B, int n,
int m){
    int k,i,j;
    if(n>m){
        k=n;
        for(i=0;i<m;i++)
            C[i]=A[i]+B[i];
        for(i=m; i<n; i++)
            C[i]=A[i];
    }
    else {
        k=m;
        for(i=0;i<n;i++)
            C[i]=A[i]+B[i];
        for(i=n; i<m; i++)
            C[i]=B[i];
    }
    printf("\n Da thuc tong:");
    In_Dathuc(C,k);
}

```

```

void Hieu_Dathuc(float *C,float *A, float *B, int n, int
m) {
    int k,i,j;
    if(n>m){
        k=n;

```

```
for(i=0;i<m;i++)
```



```
C[i]=A[i]-B[i];
for(i=m; i<n; i++)
C[i]=A[i];
}
else {
k=m;
for(i=0;i<n;i++)
C[i]=A[i]-B[i];
for(i=n; i<m; i++)
C[i]=B[i];
}
printf("\n Da thuc
hieu:");
In_Dathuc(C,k);
}

void
Thuc_Hien(void){
float *A, *B, *C;
int n, m, control
= 0; char phim;
printf("\n
Bac cua da
thuc
```

```
A:");scanf("%d", &n); printf("\n Bac cua da thuc
B:");scanf("%d", &m); n=n+1;m=m+1;
A = malloc( n * sizeof(float));
B = malloc( m * sizeof(float));
C = malloc( (m + n) * sizeof(float));
do {
clrscr();
printf("\n Tập thao tác trên đa thức");
printf("\n F1- Tạo lập đa thức");
printf("\n F2- Tính giá trị đa thức");
```

```
printf("\n F3- Tổng hai đa thức");  
printf("\n F4- Hiệu hai đa thức");  
printf("\n F10- Tạo lập đa thức");  
phim = return 0;  
switch(phim) {  
    case F1: control = 1;
```



```
        Init_Dathuc(A, n, 'A');  
        Init_Dathuc(B, m, 'B'); break;  
case F2:  
    if(control) {  
        Val_Dathuc(A,n);  
        Val_Dathuc(B, m);  
    }  
    break;
```

```

        case F3:
            if(control)
                Tong_Dathuc(C, A, B, n , m);
            break;
        case F4:
            if(control)
                Hieu_Dathuc(C, A, B, n , m);
            break;
    }
} while( phim !=F10);
free(A); free(B); free(C);
}
int main(){
    Thuc_Hien();
    return 0;
}

```

Con trỏ với mảng nhiều chiều

C không hạn chế số chiều của mảng nhiều chiều mặc dầu trong thực tế có khuynh hướng là ít sử dụng chúng hơn là mảng con trỏ. Trong mục này ta sẽ đưa ra mối liên hệ giữa con trỏ và mảng nhiều.