Giáo trình Lập trình căn bản

Lâm Hoài Bảo, Dương Văn Hiếu, Nguyễn Văn Linh

Khoa Công Nghệ Thông Tin & Truyền Thông, Đại Học Cần Thơ

2005

MỤC LỤC

MỞ ĐẦU

I. MỤC ĐÍCH

Quyển sách này cung cấp cho người đọc những kiến thức cơ bản về lập trình thông qua ngôn ngữ lập trình C. Đây là cơ sở để người đọc từng bước tiếp cận với thế giới lập trình, từ đó người đọc có một nền tảng vững chắc để có thể tiếp thu hầu hết các lĩnh vực trong chuyên ngành Công Nghệ Thông Tin.

Các vấn đề chính được trình bày trong quyển sách này:

- Các khái niệm: ngôn ngữ lập trình, kiểu dữ liệu.
- Khái niệm giải thuật và biểu diễn giải thuật
- Ngôn ngữ lập trình C (sau đây gọi tắt là C):
  - ✓ Các thành phần của C.
  - ✓ Các kiểu dữ liệu trong C.
  - ✓ Các câu lệnh trong C.
  - ✓ Cách thiết kế và sử dụng các hàm trong C.
  - ✓ Một số cấu trúc dữ liệu trong C.

Chương 1

GIẢI THUẬT VÀ BIỂU DIỄN GIẢI THUẬT

Các vấn đề được trình bày trong chương này:

- Từ bài toán đến chương trình
- Giải thuật và biểu diễn giải thuật
- Một số khái niệm: kiểu dữ liệu, ngôn ngữ lập trình, chương trình dịch

Nội dung chính của chương này là giải thuật & các cách biểu diễn giải thuật. Lý do vì chương trình máy tính của một bài toán cụ thể được tạo ra từ các biểu diễn của giải thuật.

I. TỪ BÀI TOÁN ĐẾN CHƯƠNG TRÌNH

Theo [6], mọi bài toán đều có thể được diễn giải theo một sơ đồ chung: A → B

Trong đó:

- A là giả thiết, là điều kiện ban đầu.
- B là kết luận, là mục tiêu cần đạt hoặc là cái phải tìm.
- → là suy luận, là giải pháp cần thực hiện để tìm B từ cái đã biết A.

Một bài toán trên máy tính cũng mang đầy đủ các tính chất của bài toán ở trên. Trong đó :

- A là thông tin vào (đầu vào - input)
- B là thông tin ra (đầu ra - output)
- → là chương trình tạo ra từ các lệnh cơ bản của máy tính cho phép tạo B từ A.

Như vậy, việc giải một bài toán trên máy tính là việc xác định các yếu tố đầu vào, đầu ra cũng như xác định cách giải quyết bài toán bằng chương trình máy tính.

Thí dụ: Giả sử có hai bình B1 và B2 đựng hai loại chất lỏng khác nhau, chẳng hạn bình B1 đựng rượu, bình B2 đựng nước mắm. Làm thế nào để hoán đổi chất lỏng trong 02 bình, tức bình B1 đựng nước mắm, bình B2 đựng rượu?

- Đầu vào của bài toán là 02 bình B1, B2 với mỗi bình lần lượt chứa rượu và nước mắm.
- Đầu ra của bài toán là bình B1 chứa nước mắm, bình B2 chứa rượu.
- Một dãy các bước để thực hiện bài toán này là :
  - Có thêm một bình thứ ba gọi là B3.
  - *Bước 1*: Đổ rượu từ bình B1 vào bình B3.
  - *Bước 2*: Đổ nước mắm từ bình B2 sang bình B1.
  - *Bước 3*: Đổ rượu từ bình B3 sang B2.

Trong thí dụ trên, dãy các bước để thực hiện bài toán hoán đổi chất lỏng trong 02 bình chưa phải là 1 chương trình. Tuy nhiên chương trình máy tính thực hiện bài toán này được cài đặt với các lệnh được mô tả gần với cách mô tả trong dãy các bước trên. Dãy các bước trên còn được gọi là giải thuật để giải bài toán hoán đổi chất lỏng trong 02 bình trên.

II. GIẢI THUẬT

II.1. Khái niệm giải thuật

Giải thuật là một hệ thống chặt chẽ và rõ ràng các quy tắc nhằm xác định một dãy các thao tác trên những dữ liệu vào sao cho sau một số hữu hạn bước thực hiện các thao tác đó ta thu được kết quả của bài toán.

Thí dụ: Tìm ước số chung lớn nhất của 2 số nguyên a, b.

- Đầu vào: 2 số nguyên a, b.
- Đầu ra: ước số chung lớn nhất của a, b
- Giải thuật:
  - *Bước 1*: Nhập vào hai số a và b.

- *Bước 2*: So sánh 2 số a, b chọn số nhỏ nhất gán cho UCLN.
- *Bước 3*: Nếu một trong hai số a hoặc b không chia hết cho UCLN thì thực hiện bước 4, ngược lại (cả a và b đều chia hết cho UCLN) thì thực hiện bước 5.
- *Bước 4*: Giảm UCLN một đơn vị và quay lại bước 3
- *Bước 5*: In UCLN - Kết thúc.

II.2 Các đặc trưng của giải thuật

- Tính kết thúc: Giải thuật phải dừng sau một số hữu hạn bước.
- Tính xác định: Các thao tác máy tính phải thực hiện được và các máy tính khác nhau thực hiện cùng một bước của cùng một giải thuật phải cho cùng một kết quả.
- Tính phổ dụng: Giải thuật phải "vét' hết các trường hợp và áp dụng cho một loạt bài toán cùng loại.
- Tính hiệu quả: Một giải thuật được đánh giá là tốt nếu nó đạt hai tiêu chuẩn sau:
  - Thực hiện nhanh, tốn ít thời gian.
  - Tiêu phí ít tài nguyên của máy, chẳng hạn tốn ít bộ nhớ.

Giải thuật tìm UCLN nêu trên đạt tính kết thúc bởi vì qua mỗi lần thực hiện bước 4 thì UCLN sẽ giảm đi một đơn vị cho nên trong trường hợp xấu nhất thì UCLN=1, giải thuật phải dừng. Các thao tác trình bày trong các bước, máy tính đều có thể thực hiện được nên nó có tính xác định. Giải thuật này cũng đạt tính phổ dụng vì nó được dùng để tìm UCLN cho hai số nguyên dương a và b bất kỳ. Tuy nhiên tính hiệu quả của giải thuật có thể chưa cao; cụ thể là thời gian chạy máy có thể còn tốn nhiều hơn một số giải thuật khác `mà chúng ta sẽ có dịp trở lại trong các chương tiếp theo.

II.3 Ngôn ngữ biểu diễn giải thuật

Để biểu diễn giải thuật, cần phải có một tập hợp các ký hiệu dùng để biểu diễn, mỗi ký hiệu biểu diễn cho một hành động nào đó. Tập hợp các ký hiệu đó lại tạo thành ngôn ngữ biểu diễn giải thuật.

II.3.1 Ngôn ngữ tự nhiên

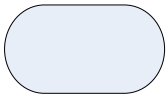
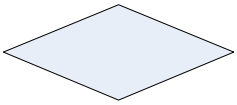
Ngôn ngữ tự nhiên là ngôn ngữ của chúng ta đang sử dụng. Chẳng hạn, các thí dụ ở trên dùng ngôn ngữ tự nhiên để biểu diễn giải thuật.

Thí dụ: Giải thuật giải phương trình bậc nhất dạng ax + b = 0 như sau:

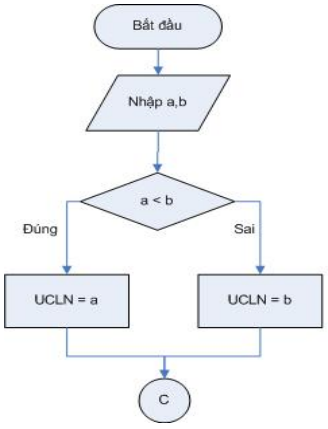
- Đầu vào: 2 số thực a, b.
- Đầu ra: Các kết luận về các trường hợp nghiệm của phương trình ax + b = 0.
- Giải thuật:
  - Bước 1: Nhận giá trị của các tham số a, b
  - Bước 2: Xét giá trị của a xem có bằng 0 hay không? Nếu a=0 thì làm bước 3, nếu a khác không thì làm bước 4.
  - Bước 3: (a bằng 0) Nếu b bằng 0 thì ta kết luận phương trình vô số nghiệm, nếu b khác 0 thì ta kết luận phương trình vô nghiệm.
  - Bước 4: (a khác 0) Kết luận phương trình có nghiệm x=-b/a.

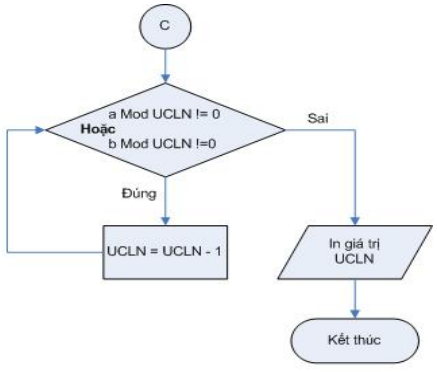
II.3.2 Lưu đồ

Lưu đồ là một tập ký hiệu trực quan dùng để thể hiện giải thuật. Các ký hiệu được sử dụng trong lưu đồ được cho trong bảng sau:

Ký hiệu	Ý nghĩa
	Bắt đầu/ Kết thúc
	Nhập/Xuất
	Tính toán (xử lý)
	Quyết định và rẽ nhánh
	Khối nối
	Đường đi

Chẳng hạn lưu đồ để biểu diễn giải thuật tìm ước số chung lớn nhất theo cách thức như trên là:





II.3.3 Một số thí dụ

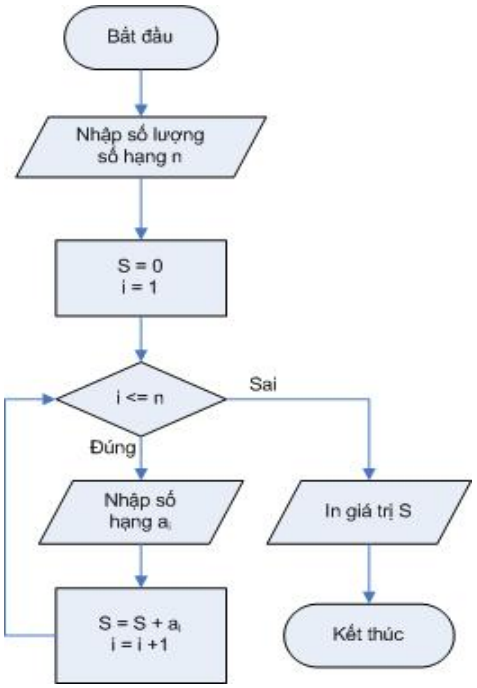
Thí dụ 1: Cần viết chương trình cho máy tính sao cho khi thực hiện chương trình đó, máy tính yêu cầu người sử dụng chương trình nhập vào các số hạng của tổng (n); nhập vào dãy các số hạng a_i của tổng. Sau đó, máy tính sẽ thực hiện việc tính tổng các số a_i này và in kết quả của tổng tính được.

Yêu cầu: Tính tổng n số $S=a_1+ a_2 + a_3 +.....+ a_n$.

Để tính tổng trên, chúng ta sử dụng phương pháp "cộng tích lũy" nghĩa là khởi đầu cho S=0. Sau mỗi lần nhận được một số hạng a_i từ bàn phím, ta cộng tích lũy a_i vào S (lấy giá trị được lưu trữ trong S, cộng thêm a_i và lưu trở lại vào S). Tiếp tục quá trình này đến khi ta tích lũy được a_n vào S thì ta có S là tổng các a_i. Chi tiết giải thuật được mô tả bằng ngôn ngữ tự nhiên như sau:

- Bước 1: Nhập số các số hạng n.
- Bước 2: Cho S=0 (lưu trữ số 0 trong S)
- Bước 3: Cho i=1 (lưu trữ số 1 trong i)
- Bước 4: Kiểm tra nếu i<=n thì thực hiện bước 5, ngược lại thực hiện bước 8.
- Bước 5: Nhập a_i
- Bước 6: Cho $S=S+a_i$ (lưu trữ giá trị $S + a_i$ trong S)
- Bước 7: Tăng i lên 1 đơn vị và quay lại bước 4.
- Bước 8: In S và kết thúc chương trình.

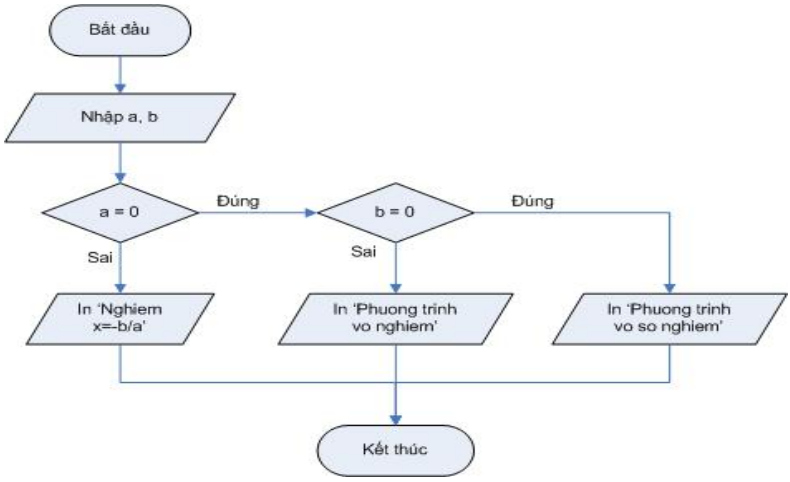
Chi tiết giải thuật trên bằng lưu đồ:



Thí dụ 2: Viết chương trình cho phép nhập vào 2 giá trị a, b mang ý nghĩa là các hệ số a, b của phương trình bậc nhất. Dựa vào các giá trị a, b đó cho biết nghiệm của phương trình bậc nhất $ax + b = 0$.

Mô tả giải thuật bằng ngôn ngữ tự nhiên (trang 6)

Lưu đồ của giải thuật giải phương trình bậc nhất:



Thí dụ 3: Viết chương trình cho phép nhập vào 1 số n, sau đó lần lượt nhập vào n giá trị a_1, a_2,…,a_n. Hãy tìm và in ra giá trị lớn nhất trong n số a_1, a_2, …, a_n.

Để giải quyết bài toán trên, chúng ta áp dụng phương pháp "thử và sửa". Ban đầu giả sử a_1 là số lớn nhất (được lưu trong giá trị max); sau đó lần lượt xét các a_i còn lại, nếu a_i nào lớn hơn giá trị max thi lúc đó max sẽ nhận giá trị là a_i. Sau khi đã xét hết các a_i thì max chính là giá trị lớn nhất cần tìm.

Mô tả giải thuật bằng ngôn ngữ tự nhiên:

- Bước 1: Nhập số n
- Bước 2: Nhập số thứ nhất a_1
- Bước 3: Gán max=a_1
- Bước 4: Gán i=2
- Bước 5: Nếu i<=n thì thực hiện bước 6, ngược lại thực hiện bước 9
- Bước 6: Nhập a_i
- Bước 7: Nếu max < a_i thì gán max=a_i.
- Bước 8: Tăng i lên một đơn vị và quay lại bước 5
- Bước 9: In max - kết thúc

Phần mô tả giải thuật bằng lưu đồ, độc giả có thể xem như 1 bài tập.

Thí dụ 4: Viết chương trình cho phép nhập vào 1 số n, sau đó lần lượt nhập vào n giá trị a_1, a_2,…,a_n. Sắp theo thứ tự tăng dần một dãy n số a_1, a_2,...a_n nói trên.

Dưới đây là một trong những phương pháp để sắp xếp thứ tự 1 dãy các số:

Giả sử ta đã nhập vào 1 dãy n số a_1, a_2,..., a_n. Việc sắp xếp dãy số này trải qua (n-1) lần:

- Lần 1: So sánh phần tử đầu tiên với tất cả các phần tử đứng sau phần tử đầu tiên. Nếu có phần tử nào nhỏ hơn phần tử đầu tiên thì đổi chỗ phần tử đầu tiên với phần tử nhỏ hơn đó. Sau lần 1, ta được phần tử đầu tiên là phần tử nhỏ nhất.

- Lần 2: So sánh phần tử thứ 2 với tất cả các phần tử đứng sau phần tử thứ 2. Nếu có phần tử nào nhỏ hơn phần tử thứ 2 thì đổi chỗ phần tử thứ 2 với phần tử nhỏ hơn đó. Sau lần 2, ta được phần tử đầu tiên và phần tử thứ 2 là đúng vị trí của nó khi sắp xếp.
- …
- Lần (n-1): So sánh phần tử thứ (n-1) với phần tử đứng sau phần tử (n-1) là phần tử thứ n. Nếu phần tử thứ n nhỏ hơn phần tử thứ (n-1) thì đổi chỗ 2 phần tử này.

Sau lần thứ (n-1), ta được danh sách gồm n phần tử được sắp thứ tự.

Mô tả giải thuật bằng ngôn ngữ tự nhiên:

- Bước 1: Gán i=1
- Bước 2: Gán j=i+1
- Bước 3: Nếu i <=n-1 thì thực hiện bước 4, ngược lại thực hiện bước 8
- Bước 4: Nếu j <=n thì thực hiện bước 5, ngược lại thì thực hiện bước 7.
- Bước 5: Nếu $a_i > a_j$ thì hoán đổi a_i và a_j cho nhau (nếu không thì thôi).
- Bước 6: Tăng j lên một đơn vị và quay lại bước 4
- Bước 7: Tăng i lên một đơn vị và quay lại bước 3
- Bước 8: In dãy số $a_1, a_2,..., a_n$ - Kết thúc.

II.4 Các cấu trúc suy luận cơ bản của giải thuật

Giải thuật được thiết kế theo các cấu trúc suy luận cơ bản sau đây:

II.4.1 Tuần tự (Sequential): Các công việc được thực hiện một cách tuần tự, công việc này nối tiếp công việc kia.

II.4.2 Lựa chọn (Selection): Lựa chọn một công việc để thực hiện căn cứ vào một điều kiện nào đó. Một số biến thể của cấu trúc này như sau:

1: Nếu < điều kiện> (đúng) thì thực hiện <công việc>

2: Nếu < điều kiện> (đúng) thì thực hiện <công việc 1>, ngược lại (điều kiện sai) thì thực hiện <công việc 2>

3: Trường hợp < i> thực hiện <công việc i>

II.4.3. Lặp (Repeating)

Thực hiện lặp lại một công việc không hoặc nhiều lần căn cứ vào một điều kiện nào đó. Có hai dạng như sau:

- *Lặp xác định*: là loại lặp mà khi mô tả giải thuật, số lần lặp của công việc đã được xác định.

- *Lặp không xác định*: là loại lặp mà khi mô tả giải thuật, số lần lặp của công việc chưa thể xác định. Tùy thuộc vào những lần thực thi khác nhau của chương trình cài đặt giải thuật, số lần lặp có thể khác nhau.

III. MỘT SỐ KHÁI NIỆM KHÁC

III.1. Ngôn ngữ lập trình

Ngôn ngữ lập trình là một ngôn ngữ dùng để viết chương trình cho máy tính. Ta có thể chia ngôn ngữ lập trình thành các loại sau: ngôn ngữ máy, hợp ngữ và ngôn ngữ cấp cao.

Ngôn ngữ máy (machine language): Là các chỉ thị dưới dạng nhị phân, can thiệp trực tiếp vào trong các mạch điện tử. Chương trình được viết bằng ngôn ngữ máy thì có thể được thực hiện ngay không cần qua bước trung gian nào. Tuy nhiên chương trình viết bằng ngôn ngữ máy dễ sai sót, cồng kềnh và khó đọc, khó hiểu vì toàn những con số 0 và 1.

Hợp ngữ (assembly language): Bao gồm tên các câu lệnh và quy tắc viết các câu lệnh đó. Tên các câu lệnh bao gồm hai phần:

phần mã lệnh (viết tựa tiếng Anh) chỉ phép toán cần thực hiện và địa chỉ chứa toán hạng của phép toán đó. Thí dụ:

```
INPUT a ; Nhập giá trị cho a từ bàn phím
LOAD a ; Đọc giá trị a vào thanh ghi tổng A
PRINT a; Hiển thị giá trị của a ra màn hình.
INPUT b
ADD b; Cộng giá trị của thanh ghi tổng A với giá trị b
```

Trong các lệnh trên thì INPUT, LOAD, PRINT, ADD là các mã lệnh còn a, b là địa chỉ. Để máy thực hiện được một chương trình viết bằng hợp ngữ thì chương trình đó phải được dịch sang ngôn ngữ máy. Công cụ thực hiện việc dịch đó được gọi là Assembler.

Ngôn ngữ cấp cao (High level language): Ra đời và phát triển nhằm phản ánh cách thức người lập trình nghĩ và làm. Rất gần với ngôn ngữ con người (Anh ngữ) nhưng chính xác như ngôn ngữ toán học. Cùng với sự phát triển của các thế hệ máy tính, ngôn ngữ lập trình cấp cao cũng được phát triển rất đa dạng và phong phú, việc lập trình cho máy tính vì thế mà cũng có nhiều khuynh hướng khác nhau: lập trình cấu trúc, lập trình hướng đối tượng, lập trình logic, lập trình hàm... Một chương trình viết bằng ngôn ngữ cấp cao được gọi là chương trình nguồn (source programs). Để máy tính "hiểu" và thực hiện được các lệnh trong chương trình nguồn thì phải có một chương trình dịch để dịch chuong trình nguồn (viết bằng ngôn ngữ cấp cao) thành dạng chương trình có khả năng thực thi.

III.2 Chương trình dịch

Muốn chuyển từ chương trình nguồn sang chương trình đích phải có chương trình dịch. Nói chung các chương trình dịch hoạt động theo 1 trong 2 cơ chế:

Thông dịch (interpreter): Là cách dịch từng lệnh một, dịch tới đâu thực hiện tới đó. Chẳng hạn ngôn ngữ LISP sử dụng trình thông dịch.

Biên dịch (compiler): Dịch toàn bộ chương trình nguồn thành chương trình đích rồi sau đó mới thực hiện. Các ngôn ngữ sử dụng trình biên dịch như Pascal, C...

Giữa thông dịch và biên dịch có khác nhau ở chỗ: Do thông dịch là vừa dịch vừa thực thi chương trình còn biên dịch là dịch xong toàn bộ chương trình rồi mới thực thi nên chương trình viết bằng ngôn ngữ biên dịch thực hiện nhanh hơn chương trình viết bằng ngôn ngữ thông dịch.

Một số ngôn ngữ ở đó chương trình dịch sử dụng kết hợp giữa thông dịch và biên dịch chẳng hạn như Java. Chương trình nguồn của Java được biên dịch tạo thành một chương trình đối tượng (một dạng mã trung gian) và khi thực hiện thì từng lệnh trong chương trình đối tượng được thông dịch thành mã máy.

III.3 Kiểu dữ liệu

Các số liệu lưu trữ trong máy tính gọi là *dữ liệu* (data). Mỗi đơn vị dữ liệu thuộc một kiểu dữ liệu nào đó.

Kiểu dữ liệu là một tập hợp các giá trị có cùng một tính chất và tập hợp các phép toán thao tác trên các giá trị đó. Người ta chia kiểu dữ liệu ra làm 2 loại: kiểu dữ liệu sơ cấp và kiểu dữ liệu có cấu trúc.

III.3.1 Kiểu dữ liệu sơ cấp

Kiểu dữ liệu sơ cấp là kiểu dữ liệu mà giá trị của nó là đơn nhất.

Thí dụ: Trong ngôn ngữ lập trình C, kiểu int gọi là kiểu sơ cấp vì kiểu này bao gồm các số nguyên từ -32768 đến 32767 và các phép toán +, -, *, /, %…

III.3.2 Kiểu dữ liệu có cấu trúc

Kiểu dữ liệu có cấu trúc là kiểu dữ liệu mà các giá trị của nó là sự kết hợp của các giá trị khác.

Thí dụ: Kiểu chuỗi ký tự trong ngôn ngữ lập trình C là một kiểu dữ liệu có cấu trúc.

Các ngôn ngữ lập trình đều có những kiểu dữ liệu do ngôn ngữ xây dựng sẵn, mà ta gọi là các kiểu chuẩn. Chẳng hạn như kiểu

int, char… trong C; integer, array… trong Pascal. Ngoài ra, hầu hết các ngôn ngữ đều cung cấp cơ chế cho phép người lập trình định nghĩa kiểu của riêng mình để phục vụ cho việc viết chương trình.

IV. BÀI TẬP

Bằng ngôn ngữ tự nhiên và lưu đồ, anh (chị) hãy mô tả giải thuật cho các bài toán sau:

1. Tính diện tích của 1 tam giác theo công thức Hê-rông:

$S = \sqrt{p*(p-a)*(p-b)*(p-c)}$

Với a, b, c là chiều dài 3 cạnh được nhập từ bàn phím.

p: Nửa chu vi.

2. Giải phương trình bậc 2 dạng $ax^2 + bx + c = 0$ với a, b, c là các số sẽ nhập từ bàn phím.

3.Tính tổng bình phương của n số nguyên có dạng sau:

$S = a_1^2 + a_2^2 + \ldots + a_n^2$ với n và a_i (i=1..n) là các số sẽ nhập từ bàn phím.

Chương 2

TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH C

Các vấn đề được trình bày trong chương này:

- Lịch sử của ngôn ngữ lập trình C.
- Các thành phần của C.

I. SƠ LƯỢC VỀ NGÔN NGỮ LẬP TRÌNH C

C là ngôn ngữ lập trình cấp cao, được sử dụng rất phổ biến để lập trình hệ thống cùng với Assembler và phát triển các ứng dụng.

Vào những năm cuối thập kỷ 60 đầu thập kỷ 70 của thế kỷ XX, Dennish Ritchie (làm việc tại phòng thí nghiệm Bell) đã phát triển ngôn ngữ lập trình C dựa trên ngôn ngữ BCPL (do Martin Richards đưa ra vào năm 1967) và ngôn ngữ B (do Ken Thompson phát triển từ ngôn ngữ BCPL vào năm 1970 khi viết hệ điều hành UNIX đầu tiên trên máy PDP-7) và được cài đặt lần đầu tiên trên hệ điều hành UNIX của máy DEC PDP-11.

Năm 1978, Dennish Ritchie và B.W Kernighan đã cho xuất bản quyển "Ngôn ngữ lập trình C" và được phổ biến rộng rãi đến nay.

Lúc ban đầu, C được thiết kế nhằm lập trình trong môi trường của hệ điều hành Unix nhằm mục đích hỗ trợ cho các công việc lập trình phức tạp. Nhưng về sau, với những nhu cầu phát triển ngày một tăng của công việc lập trình, C đã vượt qua khuôn khổ của phòng thí nghiệm Bell và nhanh chóng hội nhập vào thế giới lập trình để rồi các công ty lập trình sử dụng một cách rộng rãi. Sau đó, các công ty sản xuất phần mềm lần lượt đưa ra các phiên bản hỗ trợ cho việc lập trình bằng ngôn ngữ C và chuẩn ANSI C cũng được khai sinh từ đó.

Ngôn ngữ lập trình C là một ngôn ngữ lập trình hệ thống rất mạnh và rất "mềm dẻo", có một thư viện gồm rất nhiều các hàm

(function) đã được tạo sẵn. Người lập trình có thể tận dụng các hàm này để giải quyết các bài toán mà không cần phải tạo mới. Hơn thế nữa, ngôn ngữ C hỗ trợ rất nhiều phép toán nên phù hợp cho việc giải quyết các bài toán kỹ thuật có nhiều công thức phức tạp. Ngoài ra, C cũng cho phép người lập trình tự định nghĩa thêm các kiểu dữ liệu trừu tượng khác. Tuy nhiên, điều mà người mới vừa học lập trình C thường gặp "rắc rối" là "hơi khó hiểu" do sự "mềm dẻo" của C. Dù vậy, C được phổ biến khá rộng rãi và đã trở thành một công cụ lập trình khá mạnh, được sử dụng như là một ngôn ngữ lập trình chủ yếu trong việc xây dựng những phần mềm hiện nay.

Ngôn ngữ C có những đặc điểm cơ bản sau:

- *Tính cô đọng (compact)*: C chỉ có 32 từ khóa chuẩn và 40 toán tử chuẩn, nhưng hầu hết đều được biểu diễn bằng những chuỗi ký tự ngắn gọn.
- *Tính cấu trúc (structured)*: C có một tập hợp những chỉ thị của lập trình như cấu trúc lựa chọn, lặp… Từ đó các chương trình viết bằng C được tổ chức rõ ràng, dễ hiểu.
- *Tính tương thích (compatible)*: C có bộ tiền xử lý và một thư viện chuẩn vô cùng phong phú nên khi chuyển từ máy tính này sang máy tính khác các chương trình viết bằng C vẫn hoàn toàn tương thích.
- *Tính mềm dẻo (flexible)*: C là một ngôn ngữ rất uyển chuyển về cú pháp, chấp nhận nhiều cách thể hiện, có thể thu gọn kích thước của các mã lệnh làm chương trình chạy nhanh hơn.
- *Biên dịch (compile)*: C cho phép biên dịch nhiều tập tin chương trình riêng rẽ thành các tập tin đối tượng (object) và liên kết (link) các đối tượng đó lại với nhau thành một chương trình có thể thực thi được (executable) thống nhất.

II. BỘ CHỮ VIẾT TRONG C

Bộ chữ viết trong ngôn ngữ C bao gồm những ký tự, ký hiệu sau: (*phân biệt chữ in hoa và in thường*):

- Các ký tự hoa A, B, C...Z
- Các ký tự thường a, b, c ...z.

- Các ký số 0, 1, 2...9.
- Các ký hiệu toán học: +, -, *, /, =, <, >, (,)
- Các ký hiệu đặc biệt: :. , ; " ' _ @ # $! ^ [ ] { } ...
- Dấu cách hay khoảng trống.

III. CÁC TỪ KHÓA TRONG C

Từ khóa là các từ dành riêng (reserved words) của C mà người lập trình có thể sử dụng chúng trong chương trình tùy theo ý nghĩa của từng từ. Dưới đây là bảng liệt kê các từ khóa thông dụng của C để bạn đọc lướt qua cho có khái niệm, không nhất thiết phải nhớ ngay:

break	**case**	**const**	**continue**
default	**do**	**else**	**for**
goto	**if**	**return**	**sizeof**
struct	**typedef**	**while**	

IV. TÊN VÀ QUY CÁCH ĐẶT TÊN

Một chương trình nguồn C sử dụng khá nhiều tên hay còn gọi là định danh (identifier) như: tên hàm, tên biến, tên hằng… Mọi tên đều phải được khai báo trước khi sử dụng.

Tên có hai loại là tên chuẩn và tên do người lập trình đặt.

- Tên chuẩn là tên do C đặt sẵn như sin, cos, printf...
- Tên do người lập trình tự đặt để dùng trong chương trình của mình. Sử dụng bộ chữ cái, chữ số và dấu gạch dưới (_) để đặt tên, nhưng phải tuân thủ quy tắc:
  - Bắt đầu bằng một chữ cái hoặc dấu gạch dưới.
  - Không có khoảng trống ở giữa tên.
  - Không được trùng với từ khóa.
  - Không cấm việc đặt tên trùng với tên chuẩn nhưng khi đó ý nghĩa của tên chuẩn không còn giá trị nữa.

Thí dụ: Tên do hợp lệ: Chieu_dai, Chieu_Rong, Chu_Vi.

Tên không hợp lệ: Do Dai, 12A2,…

Chú ý: C là ngôn ngữ phân biệt ký tự hoa và ký tự thường, vì thế pi và Pi là 2 tên khác nhau.

V. CHÚ THÍCH

Khi viết chương trình đôi lúc ta cần phải có vài lời ghi chú về 1 đoạn chương trình nào đó để dễ nhớ và làm sáng sủa chương trình. Để ý rằng phần nội dung ghi chú này phải không thuộc về chương trình (khi biên dịch phần này bị bỏ qua). Phần ghi chú như vậy được gọi là chú thích. Trong ngôn ngữ lập trình C, nội dung chú thích phải được viết trong cặp dấu /* và */ (nếu chú thích trên nhiều dòng), hoặc đặt sau cặp dấu // (nếu chú thích trên 1 dòng).

Thí dụ: Chương trình hiển thị lên màn hình câu thông báo 'Hello World'.

```
#include<stdio.h>
#include<conio.h>
main (){
     /*Xuat chuoi ra man hinh*/
     printf("Hello World");
     //Cho phim bat ky
     getch();
}
```

Ý nghĩa các dòng trong đoạn chương trình trên:

- Các dòng

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

cho phép gộp các tập tin tiêu đề stdio.h và conio.h vào tập tin chương trình nguồn. Tập tin stdio.h chứa định nghĩa về các hàm xuất nhập chuẩn (chẳng hạn printf). Tập tin conio.h chứa các hàm xuất nhập như getch().

#include là chỉ thị hướng dẫn dịch của C, chỉ thị này thông báo cho chương trình dịch nạp tập tin tiêu đề stdio.h (và conio.h) vào để dịch.

- Khối

```
main()
{
     …
}
```

gọi là thân chương trình. Phần này bắt buộc phải có cho mọi chương trình C. Phần này còn được gọi là hàm main() hoặc là điểm bắt đầu thực hiện của 1 chương trình C.

- Dòng `printf("Hello World");` cho phép hiển thị giá trị của câu *Hello World* lên màn hình.
- Dòng `getch();` chờ nhận 1 ký tự từ bàn phím. Chương trình sẽ thực hiện tiếp tục sau khi một phím bất kỳ được ấn.

Phần nằm trong /* */ hoặc dòng nằm sau // là chú thích

VI. CÁC KIỂU DỮ LIỆU SƠ CẤP CHUẨN TRONG C

Các kiểu dữ liệu sơ cấp chuẩn trong C có thể được chia làm 2 dạng : kiểu số nguyên, kiểu số thực. Phần này chỉ trình bày miền giá trị của các kiểu, các phép toán trên các giá trị của các kiểu sẽ được trình bày ở các phần tiếp theo.

VI.1. Kiểu số nguyên

Kiểu số nguyên là một kiểu dữ liệu mà tập các giá trị của nó là tập con của tập số nguyên trong toán học. Tùy thuộc vào kích thước lưu trữ mà kiểu số nguyên được chia làm 3 loại: 1 byte, 2 bytes hay 4 bytes. Ứng với mỗi loại, người ta còn định nghĩa sẵn một số kiểu khác nhau. Tùy thuộc vào ứng dụng của chương trình mà người lập trình sẽ sử dụng 1 kiểu thích hợp.

VI.1.1. Kiểu số nguyên 1 byte (8 bits)

STT	Kiểu dữ liệu	Miền giá trị (Domain)
1	char	Từ **-128** đến **127**
2	unsigned char	Từ **0** đến **255**

Kiểu số nguyên một byte gồm có 2 kiểu sau:

- Kiểu char: các số nguyên thuộc kiểu **char** có giá trị từ -128 đến 127.

- Kiểu unsgined char: các số nguyên thuộc kiểu **unsigned char** có giá trị từ 0 đến 255.

Chú ý: Kiểu **char** còn có thể được gọi là kiểu ký tự vì C cho phép kiểu này được sử dụng như số nguyên và ký tự.

VI.1.2. Kiểu số nguyên 2 bytes (16 bits)

Kiểu số nguyên 2 bytes gồm có 4 kiểu sau:

STT	Kiểu dữ liệu	Miền giá trị (Domain)
1	int	Từ -32,768 đến 32,767
2	short	Từ -32,768 đến 32,767
3	unsigned int	Từ 0 đến 65,535
4	unsigned short	Từ 0 đến 65,535

VI.1.3. Kiểu số nguyên 4 byte (32 bits)

Kiểu số nguyên 4 bytes hay còn gọi là số nguyên dài (long) gồm có 2 kiểu sau:

STT	Kiểu dữ liệu	Miền giá trị (Domain)
1	long	Từ **-2,147,483,648** đến **2,147,483,647**
2	unsigned long	Từ **0** đến **4,294,967,295**

VI.2. Kiểu số thực

Kiểu số thực là một kiểu dữ liệu mà tập các giá trị của nó là tập hợp con của tập các số thực trong toán học. Người ta định nghĩa sẵn 3 kiểu số thực với kích thước, miền giá trị và độ chính xác (số chữ số thập phân) khác nhau. Tùy thuộc vào ứng dụng của chương trình mà người lập trình lựa chọn 1 kiểu thích hợp để sử dụng.

STT	Kiểu dữ liệu	Kích thước	Miền giá trị (Domain)
1	float	4 bytes	Từ **$1.2 * 10^{-38}$** đến **$3.4 * 10^{38}$.** Độ chính xác khoảng 7 chữ số
2	double	8 bytes	Từ **$2.2 * 10^{-308}$** đến **$3.4 * 10^{308}$.** Độ chính xác khoảng 15 chữ số
3	long double	10 bytes	Từ **$3.4 * 10^{-4932}$** đến **$3.4 * 10^{4932}$.** Độ chính xác khoảng 19 chữ số.

Ngoài ra ta còn có kiểu dữ liệu **void**, kiểu này mang ý nghĩa là kiểu rỗng không chứa giá trị gì cả.

VII. HẰNG

Hằng là một đại lượng mà giá trị của nó không thay đổi trong suốt quá trình thực hiện chương trình. Hằng có thể được đặt tên (theo quy tắc đặt tên trong mục IV) hoặc là một hằng trực tiếp.

Hằng có thể là một chuỗi ký tự, một ký tự, một con số xác định. Chúng có thể được biểu diễn hay định dạng (Format) với nhiều dạng thức khác nhau.

VII.2.1 Hằng số thực

Là một số thực thuộc miền giá trị của một trong 3 kiểu số thực được trình bày ở trên. Các giá trị số thực được biểu diễn theo 2 cách sau:

- *Cách 1*: Sử dụng cách viết thông thường gồm 2 phần: phần nguyên và phần thập phân; mỗi phần phân cách nhau bởi dấu chấm (.).

Thí dụ: 0.0; 123.34; -223.333; 3.00; -56.0

- *Cách 2*: Sử dụng cách viết theo số mũ hay số khoa học. Một số thực được tách làm 2 phần, cách nhau bằng ký tự e hay E

Phần giá trị: là một số nguyên hay số thực được viết theo cách 1.

Phần mũ: là một số nguyên

→ Giá trị của số thực là: *Phần giá trị nhân với 10 mũ phần mũ*.

Thí dụ: 1234.56e-3 = 1.23456 (là số $1234.56 * 10^{-3}$)

-123.45E4 = -1234500 (là $-123.45 * 10^4$)

VI.2.2 Hằng số nguyên

Một hằng số nguyên là một số nguyên thuộc miền giá trị của các số nguyên 1 byte, 2 bytes và 4 bytes.

- *Hằng số nguyên hệ thập phân*: là số nguyên có giá trị thuộc kiểu số nguyên 1 byte/2 bytes/4 bytes và được viết như cách viết thông thường trong toán học (sử dụng các ký số từ 0 đến 9 để biểu diễn một giá trị nguyên).

Thí dụ: 123 (một trăm hai mươi ba),

-242 (trừ hai trăm bốn mươi hai).

- *Hằng số nguyên hệ bát phân*: là số nguyên có giá trị thuộc kiểu số nguyên 1 byte/2 bytes/4 bytes và được biểu diễn bởi 8 chữ số từ 0 đến 7, bắt đầu bằng số 0.

Cách biểu diễn: 0<các ký số từ 0 đến 7>

Thí dụ: 0345 (số 345 trong hệ bát phân)

-020 (số -20 trong hệ bát phân)

Cách tính giá trị thập phân của số bát phân như sau:

Số bát phân : $0d_nd_{n-1}d_{n-2}\ldots d_1d_0$ (d_i có giá trị từ 0 đến 7)

=> Giá trị thập phân= $\sum_{i=0}^{n} d_i * 8^i$

0345=229, 020=16

- *Hằng số nguyên hệ thập lục phân*: là số nguyên có giá trị thuộc kiểu số nguyên 1 byte/2 bytes/4 bytes và được biểu diễn bởi 10 chữ số từ 0 đến 9 và 6 ký tự A, B, C, D, E, F

Ký tự	Giá trị
A	10
B	11
C	12
D	13
E	14
F	15

Cách biểu diễn:

0x<các ký số từ 0 đến 9 và 6 ký tự từ A đến F>

Thí dụ:

0x345 (số 345 trong hệ 16)
0x20 (số 20 trong hệ 16)
0x2A9 (số 2A9 trong hệ 16)

Cách tính giá trị thập phân của số thập lục phân:

Số thập lục phân : $0xd_nd_{n-1}d_{n-2}\ldots d_1d_0$
(d_i từ 0 đến 9 hoặc A đến F)

=> Giá trị thập phân= $\sum_{i=0}^{n} d_i * 16^i$

Thí dụ: 0x345=827 , 0x20=32 , 0x2A9= 681

- *Hằng số nguyên có định trước kiểu*: Đôi khi ta muốn ghi các hằng số với kiểu được định trước tường minh, điều này có thể được thực hiện bằng cách thêm một (một số) ký tự vào cuối dãy số.

- U (hoặc u): cho kiểu unsigned int
- L (hoặc l): cho kiểu long
- UL (hoặc ul): cho kiểu unsigned long

Thí dụ:

65000U: hằng số nguyên kiểu unsigned int.
123456789L: hằng số nguyên kiểu long

VI.2.3. Hằng ký tự

Hằng ký tự là một ký tự riêng biệt được viết trong cặp dấu nháy đơn (‘).

Thí dụ: ‘a’, ‘A’, ‘0’, ‘9’

Mỗi hằng ký tự được lưu trữ đúng 1 byte trong bộ nhớ.

Một giá trị hằng ký tự là một phần tử của 1 tập hữu hạn các ký tự được sắp thứ tự. Các máy tính đều sử dụng tập các ký tự như vậy để trao đổi thông tin với nhau qua các thiết bị xuất nhập. Có nhiều cách sắp xếp bộ chữ khác nhau và có một bộ chữ (bộ mã) được sử dụng phổ biến để trao đổi thông tin giữa các thiết bị, nhất là trên máy tính. Đó là bộ mã ASCII (American Standard Code for Information Interchange).

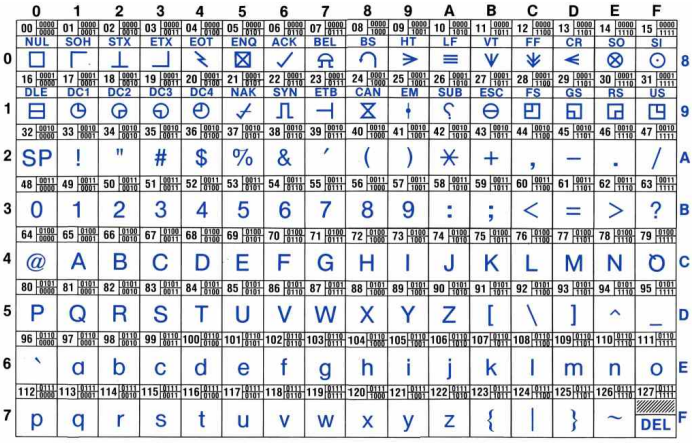
Mỗi ký tự được mã hóa đúng bằng 1 byte, vì thế bảng mã ASCII có thể mã hóa tới 256 ký tự (2^8). Cách thức mã hóa được thực hiện bằng cách gán cho mỗi ký tự một giá trị số thuộc [0..255] gọi là mã ASCII của ký tự đó.

Thí dụ: Ký tự ‘A’, ‘B’ có mã ASCII lần lượt là 65, 66

Ký tự ‘a’, ‘b’ có mã ASCII lần lượt là 97, 98.

Thực tế chỉ có 128 giá trị số đầu tiên được sử dụng để mã hóa các ký tự thông thường, 128 giá trị tiếp theo (từ 128 - 255) được sử dụng để mã hóa cho các ký tự riêng của 1 số ngôn ngữ, các ký tự toán học…

Bảng dưới đây là bảng mã ASCII của 128 ký tự đầu tiên (còn được gọi là bảng mã ASCII chuẩn):



Chúng ta có thể thực hiện các phép toán số học trên 2 ký tự (thực chất là thực hiện phép toán trên giá trị ASCII của chúng)

VI.2.4. Hằng chuỗi ký tự

Hằng chuỗi ký tự là một chuỗi hay một xâu ký tự được đặt trong cặp dấu nháy kép (“).

Thí dụ: “Ngon ngu lap trinh C”, “Khoa CNTT-DHCT”

Chú ý:

1. Một chuỗi không có nội dung “” được gọi là chuỗi rỗng.

2. Khi lưu trữ trong bộ nhớ, một chuỗi được kết thúc bằng ký tự NULL (‘\0’: mã ASCII là 0).

3. Để biểu diễn ký tự đặc biệt bên trong chuỗi ta phải thêm dấu \ phía trước.

Thí dụ:

“I’m a student” phải viết “I\’m a student”

Một số ký tự đặc biệt

Ký tự	Giá trị thập lục phân	Ký tự được hiển thị	Ý nghĩa
\a	0x07	BEL	Phát ra tiếng chuông
\b	0x08	BS	Di chuyển con trỏ sang trái 1 ký tự và xóa ký tự bên trái (backspace)
\f	0x0C	FF	Sang trang
\n	0x0A	LF	Xuống dòng
\r	0x0D	CR	Trở về đầu dòng
\t	0x09	HT	Tab theo cột (giống gõ phím Tab)
\\	0x5C	\	Dấu \
\’	0x2C	‘	Dấu nháy đơn (‘)
\”	0x22	“	Dấu nháy kép (“)
\?	0x3F	?	Đấu chấm hỏi (?)
\0	0x00		Ký tự NULL (rỗng)

VII. BIẾN

VII.1. Biến

Biến là một đại lượng được người lập trình định nghĩa và được đặt tên thông qua việc khai báo biến. Biến dùng để lưu trữ giá trị dữ liệu trong quá trình thực hiện chương trình và giá trị của biến có thể bị thay đổi trong quá trình này. Việc đặt tên biến phải tuân theo quy tắc đặt tên trong mục IV.

Khi khai báo biến ta phải xác định kiểu cho nó. Và khi đó, giá trị của biến phải thuộc miền giá trị của kiểu.

VII.1.1. Cú pháp khai báo biến:

<Kiểu dữ liệu> Danh sách các tên biến cách nhau bởi dấu phẩy;

Thí dụ:

```
int   a, b, c; // Ba biến a, b,c có kiểu int
long n;        // Biến n có kiểu long
float    nua_chu_vi;      /*Biến nua_chu_vi
                          có kiểu float*/
double  dien_tich;        /*Biến dien_tich có
                          kiểu double*/
```

Lưu ý: Phải có dấu chấm phẩy ở cuối phần khai báo biến.

VII.1.2. Vị trí khai báo biến trong C

Trong ngôn ngữ lập trình C, ta phải khai báo biến đúng vị trí. Nếu khai báo (đặt các biến) không đúng vị trí sẽ dẫn đến những sai sót ngoài ý muốn mà người lập trình không lường trước (hiệu ứng lề). Về cơ bản, ta có 2 cách đặt vị trí của biến như sau:

a) *Khai báo biến ngoài*: Các biến này được đặt bên ngoài tất cả các hàm và nó có tác dụng hay ảnh hưởng đến toàn bộ chương trình (còn gọi là biến toàn cục).

Thí dụ:

```
int      i;    /*Bien ben ngoai */
float    pi;    /*Bien ben ngoai*/
main()
{ …
}
```

b) *Khai báo biến trong*: Các biến được đặt ở bên trong hàm, hay trong một khối lệnh. Các biến này chỉ có tác dụng hay ảnh hưởng đến hàm hay khối lệnh chứa nó. Vì thế chúng còn được gọi là các biến cục bộ của hàm hay khối lệnh. Vị trí khai báo của các biến này là ở đầu mỗi khối lệnh.

Thí dụ 1:

```
#include <stdio.h>
#include<conio.h>
int bienngoai;/*khai bao bien ngoai*/
main()
{
   int j,i; // khai bao bien cuc bo trong ham main
   i=1; j=2;
   bienngoai=3;
   printf("\n Gia7 tri cua i la %d",i);
        /*%d : In một số nguyên*/
   printf("\n Gia tri cua j la %d",j);
   printf("\n Gia tri cua bienngoai la
                  %d",bienngoai);
   getch();
}
```

Thí dụ 2:

```
#include <stdio.h>
#include<conio.h>
main ()
{  int i, j;       /*Bien cuc bo*/
   i=4; j=5;
   printf("\n Gia tri cua i la %d",i);
   printf("\n Gia tri cua j la %d",j);
   if(j>i)
   {
        int  hieu=j-i;  /* Bien cuc bo */
        printf("\n j tru i la %d",hieu);
   }
   else
```

```
        {
              int hieu=i-j; /*Bien cuc bo*/
              printf("\n i tru j la %d",hieu);
        }
        getch();
    }
```

VIII. BIỂU THỨC

Biểu thức là một sự kết hợp giữa các toán tử (operator) và các toán hạng (operand) theo đúng một trật tự nhất định.

Mỗi toán hạng có thể là một hằng, một biến, một hàm hoặc một biểu thức khác.

Mỗi biểu thức có một giá trị. Giá trị của 1 biểu thức có được bằng cách áp dụng toán tử lên các toán hạng.

Trong một biểu thức có nhiều toán tử có cùng độ ưu tiên thì thứ tự thực hiện các phép toán là từ trái sang phải (*quy tắc kết hợp trái*).

Trong một biểu thức có nhiều toán tử có độ ưu tiên khác nhau thì toán tử nào có độ ưu tiên cao hơn sẽ được thực hiện trước (*quy tắc ưu tiên*).

Nếu cần thay đổi thứ tự thực hiện các toán tử theo 2 quy tắc trên, ta dùng cặp dấu ngoặc đơn () để chỉ định toán tử nào sẽ được thực hiện trước.

Thí dụ: Biểu thức nghiệm của phương trình bậc hai:

(-b + sqrt(Delta))/(2*a)

Trong đó

- 2 là hằng; a, b, Delta là biến.

- sqrt: hàm tính căn bậc 2

- +, *, /, - là các toán tử

- Biểu thức có sử dụng các dấu ngoặc đơn để tăng độ ưu tiên.

VIII.1 Các toán tử số học

Các toán tử 2 ngôi: gồm có +, -, *, /, %.

Toán tử	Miền giá trị của toán hạng	Miền giá trị của kết quả	Ý nghĩa
+	Nguyên, Thực	Nguyên, Thực	Cộng 2 số
-	Nguyên, Thực	Nguyên, Thực	Trừ 2 số
*	Nguyên, Thực	Nguyên, Thực	Nhân 2 số
/	Nguyên	Nguyên	Chia lấy phần nguyên
%	Thực	Thực	Chia 2 số, kết quả là một số thực

Tăng và giảm (++ & --)

Đây là 2 toán tử 1 ngôi để làm tăng (++) hoặc giảm (--) giá trị của biến.

Chẳng hạn:

++x giống như x = x + 1

x-- giống như x = x – 1

Cả 2 toán tử tăng và giảm đều có thể tiền tố (đặt trước) hay hậu tố (đặt sau) toán hạng. Thí dụ: x = x + 1 có thể viết x++ (hay ++x)

Tuy nhiên giữa tiền tố và hậu tố có sự khác biệt khi sử dụng trong 1 biểu thức.

Đối với toán tử ++, cả 2 trường hợp đều làm tăng giá trị của biến. Nhưng ++x sẽ làm tăng giá trị của x lên 1 đơn vị trước khi giá trị của x được sử dụng trong khi x++ sẽ sử dụng giá trị của x trước, sau đó x mới tăng giá trị lên 1 đơn vị.

Thí dụ:

```
x = 10
y = ++x  //y = 11
```

Tuy nhiên:

```
x = 10
y = x++ //y = 10
```

Tương tự đối với toán tử --.

Thứ tự ưu tiên của các toán tử số học:

++ -- sau đó là * / % rồi mới đến + -

VIII.2 Các toán tử quan hệ và các toán tử Logic

Trong C, mọi giá trị khác 0 được xem là đúng còn 0 là sai. Các biểu thức sử dụng các toán tử quan hệ và Logic gọi là các biểu thức logic và giá trị của 1 biểu thức logic là 1 hoặc 0.

Các toán tử quan hệ

Toán tử	Miền giá trị của toán hạng	Miền giá trị của kết quả	Ý nghĩa
>	Số, ký tự	0, 1	Lớn hơn
>=	Số, ký tự	0, 1	Lớn hơn hoặc bằng
<	Số, ký tự	0, 1	Nhỏ hơn
<=	Số, ký tự	0, 1	Nhỏ hơn hoặc bằng
==	Số, ký tự	0, 1	Bằng
!=	Số, ký tự	0, 1	Khác

Các toán tử logic

Toán tử	Miền giá trị của toán hạng	Miền giá trị của kết quả	Ý nghĩa
&&	0, 1	0, 1	AND
‖	0, 1	0, 1	OR
!	0, 1	0, 1	NOT

Bảng chân trị cho các toán tử Logic:

p	q	p&&q	p‖q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Các toán tử quan hệ và Logic đều có độ ưu tiên thấp hơn các toán tử số học. Do đó một biểu thức như: 10 > 1+ 12 sẽ được xem là 10 > (1 + 12) và kết quả là sai (0).

Ta có thể kết hợp vài toán tử lại với nhau thành biểu thức như sau:

10>5&&!(10<9)||3<=4 Kết quả là đúng

Thứ tự ưu tiên của các toán tử quan hệ và Logic

Cao nhất: !

> >= < <=

== !=

&&

Thấp nhất: ||

VIII.3 Các toán tử thao tác bit

Đây là các toán tử cho phép thao tác trên từng bit nhị phân của các toán hạng là các số nguyên.

Toán tử	Ý nghĩa
&	AND theo từng bit
|	OR theo từng bit
^	XOR theo từng bit
~	Đảo bit

Thí dụ:

- Với số nguyên 3, dạng nhị phân là:

3 = 0000 0000 0000 0011

~3 = 1111 1111 1111 1100

- 3 & 5 sẽ có kết quả:

0000 0000 0000 0011 3

0000 0000 0000 0110 6

Kết quả: 0000 0000 0000 0010 4

Bảng chân trị của toán tử ^ (XOR)

p	q	p^q
0	0	0
0	1	1
1	0	1
1	1	0

Phép toán dịch trái và dịch phải:

Phép tính nhân một số nguyên với một số là lũy thừa của 2 có thể thực hiện nhanh hơn nếu dùng >> (dịch phải) hoặc << (dịch trái).

- N << M: dịch sang trái số nguyên N đi M bit, tương đương $N*2^M$.

- N >> M: dịch sang phải số nguyên N đi M bit, tương đương $N/2^M$.

Thí dụ: 4<<3, kết quả là 32 ($4*2^3$).

VIII.4 Toán tử ? cùng với :

C có một toán tử rất mạnh và trong 1 số trường hợp có thể dùng để thay thế cho các câu lệnh của If-Then-Else. Cú pháp của việc sử dụng toán tử ? là:

E1 ? E2 : E3

Trong đó E1, E2, E3 là các biểu thức.

Ý nghĩa: Trước tiên E1 được ước lượng, nếu đúng E2 được ước lượng và nó trở thành giá trị của biểu thức; nếu E1 sai, E2 được ước lượng và trở thành giá trị của biểu thức.

Thí dụ:

```
X = 10
Y = X > 9 ? 100 : 200
```

Thì Y được gán giá trị 100, nếu X nhỏ hơn 9 thì Y sẽ nhận giá trị là 200. Đoạn mã này tương đương cấu trúc if như sau:

```
X = 10
if (X < 9) Y = 100
else Y = 200
```

*VIII.5 Toán tử con trỏ & và **

Một con trỏ là địa chỉ trong bộ nhớ của một biến. *Một biến con trỏ* là một biến được khai báo riêng để chứa một con trỏ đến một đối tượng của kiểu đã chỉ ra nó. Ta sẽ tìm hiểu kỹ hơn về con

trỏ trong chương về con trỏ. Phần này chỉ đề cập ngắn gọn đến hai toán tử thường được sử dụng để thao tác với các con trỏ.

Toán tử thứ nhất là &, là một toán tử trả về địa chỉ bộ nhớ của 1 biến.

Thí dụ:

m = &count

Đặt vào biến m địa chỉ bộ nhớ của biến count.

Chẳng hạn, biến count ở vị trí bộ nhớ 2000, giả sử count có giá trị là 100. Sau câu lệnh trên m sẽ nhận giá trị 2000.

Toán tử thứ hai là *, là một bổ sung cho &; đây là một toán tử trả về giá trị của biến được cấp phát tại địa chỉ theo sau đó.

Thí dụ:

q = *m

Sẽ đặt giá trị của count vào q. Bây giờ q sẽ có giá trị là 100 vì 100 được lưu trữ tại địa chỉ 2000.

VIII.6 Toán tử dấu phẩy ,

Toán tử dấu , được sử dụng để kết hợp các biểu thức lại với nhau. Bên trái của toán tử dấu , luôn được xem là kiểu void. Điều đó có nghĩa là biểu thức bên phải trở thành giá trị của tổng các biểu thức được phân cách bởi dấu phẩy.

Thí dụ: x = (y=3, y+1);

Trước hết gán 3 cho y rồi gán 4 cho x. Cặp dấu ngoặc đơn là cần thiết vì toán tử dấu , có độ ưu tiên thấp hơn toán tử gán.

VIII.7 Xem các dấu ngoặc đơn và cặp dấu ngoặc vuông là toán tử

Trong C, cặp dấu ngoặc đơn là toán tử và chúng được dùng để tăng độ ưu tiên của các biểu thức bên trong nó.

Các cặp dấu ngoặc vuông thực hiện thao tác truy xuất phần tử trong mảng (sẽ được trình bày rõ ràng hơn trong chương Mảng).

VIII.8 Tổng kết về độ ưu tiên

Cao nhất	() []
	! ~ ++ -- (Kiểu) * &
	* / %
	+ -
	<< >>
	< <= > >=
	&
	^
	|
	&&
	||
	?:
	= += -= *= /=
Thấp nhất	,

VIII.9 Phép gán mở rộng trong C

Trong một số trường hợp, ta có thể viết tắt như sau: x +=10. Thực chất của cách viết này là tương đương với cách viết x = x + 10.

Cách viết như trong thí dụ vừa nêu có thể thực hiện trên tất cả các toán tử hai ngôi của C. Tổng quát:

(Biến) = (Biến) (Toán tử) (Biểu thức)

có thể được viết:

(Biến) (Toán tử)= Biểu thức)

IX. CẤU TRÚC CỦA 1 CHƯƠNG TRÌNH C

IX.1. Tiền xử lý và biên dịch

Trong C, việc dịch một tập tin nguồn được tiến hành trên hai bước hoàn toàn độc lập với nhau:

- Tiền xử lý.

- Biên dịch.

Hai bước này trong phần lớn thời gian được nối tiếp với nhau một cách tự động theo cách thức mà ta có ấn tượng rằng nó đã được thực hiện như là một xử lý duy nhất. Nói chung, ta thường nói đến việc tồn tại của một bộ tiền xử lý (preprocessor) nhằm chỉ rõ chương trình thực hiện việc xử lý trước. Ngược lại, các thuật ngữ trình biên dịch hay sự biên dịch vẫn còn nhập nhằng bởi vì nó chỉ ra khi thì toàn bộ hai giai đoạn, khi thì lại là giai đoạn thứ hai.

Bước tiền xử lý tương ứng với việc cập nhật trong văn bản của chương trình nguồn, chủ yếu dựa trên việc diễn giải các mã lệnh rất đặc biệt gọi là các chỉ thị dẫn hướng của bộ tiền xử lý (destination directive of preprocessor); các chỉ thị này được nhận biết bởi chúng bắt đầu bằng ký hiệu (symbol) #.

Hai chỉ thị quan trọng nhất là:

- Chỉ thị sự gộp vào các tập tin nguồn khác: #include

- Chỉ thị việc định nghĩa các macros hoặc ký hiệu: #define

Chỉ thị đầu tiên được sử dụng trước hết là nhằm gộp vào nội dung của các tập tin cần có (tập tin thư viện). Chỉ thị này thường sử dụng vì bởi các hàm của thư viện chuẩn của C được định nghĩa trong các tập tin thư viện; do đó, muốn sử dụng các hàm này, tập tin thư viện định nghĩa chúng phải được gộp vào.

Thí dụ: #include <stdio.h>

Chỉ thị thứ hai rất hay được sử dụng trong các tập tin thư viện (header file) đã được định nghĩa trước đó và thường được khai thác bởi các lập trình viên trong việc định nghĩa các ký hiệu như là:

```
#define NB_COUPS_MAX 100
#define SIZE 25
```

IX.2 Cấu trúc một chương trình C

Một chương trình C bao gồm các phần như: Các chỉ thị tiền xử lý, khai báo biến ngoài, các hàm tự tạo, chương trình chính (hàm main).

Cấu trúc có thể như sau:

Các chỉ thị tiền xử lý (Preprocessor directives)
#include <Tên tập tin thư viện>
#define ….

Thí dụ:

```
#include<stdio.h>
#define MAXINT 32767
```

Định nghĩa kiểu dữ liệu (phần này không bắt buộc): dùng để đặt tên lại cho một kiểu dữ liệu nào đó để gợi nhớ hay đặt 1 kiểu dữ liệu cho riêng mình dựa trên các kiểu dữ liệu đã có.

Cú pháp: **typedef** <Tên kiểu cũ> <Tên kiểu mới>;

Thí dụ: typedef int SoNguyen; // Kiểu SoNguyen là kiểu int

Khai báo các prototype (tên hàm, các tham số, kiểu kết quả trả về,… của các hàm *sẽ cài đặt trong phần sau, phần này không bắt buộc*): phần này chỉ là các khai báo đầu hàm, không phải là phần định nghĩa hàm.

Khai báo các biến ngoài (các biến toàn cục) *phần này không bắt buộc*: phần này khai báo các biến toàn cục được sử dụng trong cả chương trình.

Chương trình chính (hàm main), phần này bắt buộc phải có

```
main()
{
```

Các khai báo cục bộ trong hàm main: Các khai báo này chỉ tồn tại trong hàm mà thôi, có thể là khai báo biến hay khai báo kiểu.

Các câu lệnh dùng để định nghĩa hàm main

```
}
```

Cài đặt các hàm

```
<Kiểu dữ liệu trả về> <Tên hàm>( các tham số)
{
        Các khai báo cục bộ trong hàm.
        Các câu lệnh dùng để định nghĩa hàm
        [return  <kết quả trả về>;]
}
…
```

Một chương trình C được thực thi từ hàm main (thông thường là từ câu lệnh đầu tiên đến câu lệnh cuối cùng).

IX.3 Cú pháp khai báo các phần bên trong môt chương trình C

IX.3.1. Chỉ thị #include để sử dụng tập tin thư viện

Cú pháp:

#include <Tên tập tin>

hay #include "Tên đường dẫn"

Thí dụ: #include <stdio.h>

Nếu ta dùng #include"Tên đường dẫn" thì ta phải chỉ rõ tên ở đâu, tên thư mục và tập tin thư viện.

Thí dụ: #include"C:\\TC\\math.h"

Trong trường hợp tập tin thư viện nằm trong thư mục hiện hành thì ta chỉ cần đưa tên tập tin thư viện. Thí dụ: #include"math.h".

IXI.3.2. Chỉ thị #define để định nghĩa một tên

Chỉ thị này có cú pháp dạng đơn giản như sau:

#define <Tên> <Giá trị>

Ví dụ:

#define MAXINT 32767

Trong thí dụ trên, chỉ thị define định nghĩa một tên MAXINT; tên này có thể sử dụng như một hằng nguyên có giá trị là 32767.

IX.3.3 Khai báo các prototype của hàm

Cú pháp:

<Kiểu kết quả trả về> Tên hàm (danh sách đối số)

Thí dụ:

```
long giaithua( int  n);
double  x_mu_y(float  x, float y);
```

IX.3.4. Cấu trúc của hàm main

Hàm main chính là chương trình chính, gồm các khai báo, các lệnh xử lý, các lời gọi các hàm khác.

Cú pháp:

main(đối số)
{
Các khai báo và các câu lệnh định nghĩa hàm
}

Thí dụ:

```
main()
{
   int a=5, b=6,c;
   float   x=3.5, y=4.5,z;
   printf("Day la chuong trinh chinh");
   c= a + b;
   printf("\n%d + %d = %d",a,b,c);
   z= x+y;
   printf("\n%f + %f = %f", x,y,z);
   getch();
}
```

Chương 3

CÁC CÂU LỆNH ĐƠN TRONG C

Các vấn đề được trình bày trong chương này:

- Khái niệm và phân loại câu lệnh
- Các lệnh đơn trong C :
  - Gán.
  - Nhập dữ liệu từ bàn phím.
  - Hiển thị kết quả lên màn hình.

I. CÂU LỆNH

I.1. Khái niệm câu lệnh

Một câu lệnh (statement) xác định một công việc mà chương trình phải thực hiện để xử lý dữ liệu đã được mô tả và khai báo. Trong C, các câu lệnh được ngăn cách với nhau bởi dấu chấm phẩy (;).

I.2. Phân loại

Có hai loại câu lệnh: lệnh đơn và lệnh có cấu trúc.

Lệnh đơn là một lệnh không chứa các lệnh khác. Các lệnh đơn gồm: lệnh gán, các câu lệnh nhập xuất dữ liệu…

Lệnh có cấu trúc là lệnh trong đó chứa các lệnh khác. Lệnh có cấu trúc bao gồm: cấu trúc điều kiện rẽ nhánh, cấu trúc điều kiện lựa chọn, cấu trúc lặp và cấu trúc lệnh hợp thành. Lệnh hợp thành (khối lệnh) là một nhóm bao gồm nhiều khai báo biến và các lệnh được gom vào trong cặp dấu {}.

II. CÁC LỆNH ĐƠN

II.1. Lệnh gán

Lệnh gán (assignment statement) dùng để gán giá trị của một biểu thức cho một biến.

Cú pháp: <Tên biến> = <biểu thức>

Thí dụ:

```
main() {
      float Dai,Rong, Chu_Vi;
      Dai = 10.0; // Gán 10.0 cho biến Dai
      Rong = 5.0; // Gán 5.0 cho biến Rong
      // Tính chu vi hình chữ nhật
      Chu_Vi = 2*(Dai+Rong);
}
```

Nguyên tắc khi dùng lệnh gán là kiểu của biến và kiểu của biểu thức phải giống nhau, gọi là có sự tương thích giữa các kiểu dữ liệu. Chẳng hạn thí dụ sau cho thấy một sự không tương thích về kiểu:

```
main() {
      int x;
      x = 10; // Gán hằng số 10 cho biến x
      y = "Xin chao";
      //y có kiểu int, còn "Xin chao" có kiểu char*
}
```

Khi biên dịch chương trình này, C sẽ báo lỗi "*Cannot convert 'char *' to 'int'*" tức là C không thể tự động chuyển đổi kiểu từ char * (chuỗi ký tự) sang int.

Tuy nhiên trong đa số trường hợp sự tự động biến đổi kiểu để sự tương thích về kiểu sẽ được thực hiện. *Thí dụ*:

```
main() {
      float Dai,Rong, Chu_Vi;
      Dai = 10; // Gán 10 cho biến Dai
      Rong = 5; // Gán 5 cho biến Rong
      // Tính chu vi hình chữ nhật
      Chu_Vi = 2*(Dai+Rong);
}
```

Trong thí dụ trên, 10 là 1 hằng int được gán cho biến Dai kiểu float (tương tự 5 được gán cho Rong). Ở đây có một sự chuyển đổi kiểu tự động từ int sang float để cuối cùng biến Dai (Rong) lưu những giá trị là các số float.

Trong nhiều trường hợp để tạo ra sự tương thích về kiểu, ta phải sử dụng đến cách thức chuyển đổi kiểu một cách tường minh. Cú pháp của phép toán này như sau:

(Tên kiểu) <Biểu thức>

→ *Ý nghĩa : Chuyển đổi kiểu của <Biểu thức> thành kiểu mới <Tên kiểu>. Chẳng hạn như:*

```
float f;
f = (float) 10 / 4; // f lúc này là 2.5
```

Chú ý:

- Khi một biểu thức được gán cho một biến thì giá trị của nó sẽ thay thế giá trị cũ mà biến đã lưu giữ trước đó.

- Trong câu lệnh gán, dấu = là một toán tử; do đó nó có thể được sử dụng là một thành phần của biểu thức. Trong trường hợp này giá trị của biểu thức gán chính là giá trị của biến.

Thí dụ:

```
int x, y;
y = (x = 3, x + 1); // y lúc này là 4
```

- Ta có thể gán trị cho biến lúc biến được khai báo theo cách thức sau:

<Tên kiểu> <Tên biến> = <Biểu thức>;

Thí dụ:
```
float Dai = 10.0, Rong=5.0;
```

II.2. Nhập giá trị từ bàn phím cho biến (hàm scanf())

Thực chất đây là việc đọc dữ liệu từ bàn phím và gán cho các biến trong chương trình khi chương trình thực thi. Để thực hiện được việc này, ngôn ngữ C hỗ trợ hàm scanf được định nghĩa trong thư viện ***stdio.h***.

Cú pháp:

scanf("Chuỗi định dạng", Địa chỉ của các biến);

Giải thích:

- *Chuỗi định dạng*: dùng để quy định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân... Bảng dưới đây là một số định dạng khi nhập biến thuộc kiểu số nguyên, số thực, ký tự,…:

Định dạng	**Ý nghĩa**
%[số ký số]d	Nhập số nguyên có tối đa <số ký số>
%[số ký số] f	Nhập số thực có tối đa <số ký số> tính cả dấu chấm
%c	Nhập một ký tự
Thí dụ:	
%d	Nhập số nguyên
%4d	Nhập số nguyên tối đa 4 ký số, nếu nhập nhiều hơn 4 ký số thì chỉ nhận được 4 ký số đầu tiên
%f	Nhập số thực
%6f	Nhập số thực tối đa 6 ký số (tính luôn dấu chấm thập phân), nếu nhập nhiều hơn 6 ký số thì chỉ nhận được 6 ký số đầu tiên (hoặc 5 ký số với dấu chấm)

- *Địa chỉ của các biến*: là *địa chỉ của các biến* mà ta cần nhập giá trị cho biến đó. Thông thường các địa chỉ này được viết như sau: **&<tên biến>**.

Thí dụ:

```
//Doc gia tri cho bien1 co kieu nguyen
scanf("%d",&bien1);
//Doc gia tri cho bien2 co kieu thực
scanf("%f",&bien2);
//Doc gia tri cho Dai và Rong co thuc
scanf("%f%f",&Dai,&Rong);
//bien3 nhận giá trị là 1 ký tự
scanf("%c",&bien3);
```

Lưu ý:

- Chuỗi định dạng phải đặt trong cặp dấu nháy kép (“”).
- Các địa chỉ biến phải cách nhau bởi dấu phẩy (,).
- Có bao nhiêu biến thì phải có bấy nhiêu định dạng.
- Thứ tự của các định dạng phải phù hợp với thứ tự của các biến.
- Để nhập giá trị ký tự (hoặc chuỗi ký tự) được chính xác, hàm ***fflush(stdin)*** nên được sử dụng để loại bỏ các ký tự còn nằm trong vùng đệm bàn phím trước khi sử dụng hàm scanf().

- Để nhập vào một chuỗi ký tự (không chứa khoảng trắng hay *kết thúc bằng khoảng trắng*), chúng ta có thể khai báo kiểu *mảng ký tự*, sử dụng định dạng %s và *tên biến thay cho địa chỉ biến*.
- Để đọc vào một chuỗi ký tự có chứa khoảng trắng (*kết thúc bằng phím Enter)* thì *phải dùng hàm gets*().

Ví dụ:

```
int   biennguyen;
float  bienthuc;
char   bienchar;
char  chuoi1[20], chuoi2[20];
```

Nhập giá trị cho các biến:

```
scanf("%3d",&biennguyen);
```

Nếu ta nhập 1234455 thì giá trị của biennguyen là 3 ký số đầu tiên (123). Các ký số còn lại sẽ còn nằm lại trong vùng đệm.

```
scanf("%5f",&bienthuc);
```

Nếu ta nhập 123.446 thì giá trị của bienthuc là 123.4, các ký số còn lại sẽ còn nằm trong vùng đệm.

```
scanf("%2d%5f",&biennguyen, &bienthuc);
```

Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng như sau: 1223 3.142325

- 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.

- 2 ký số tiếp theo trước khoảng trắng (23) sẽ được đọc vào cho bienthuc.

```
scanf("%2d%5f%c",&biennguyen,&bienthuc,
                        &bienchar) ;
```

Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng như sau: 12345 3.142325:

- 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.

- 3 ký số tiếp theo trước khoảng trắng (345) sẽ được đọc vào cho bienthuc.

- Khoảng trắng sẽ được đọc cho bienchar.

Nếu ta chỉ nhập 1 số gồm nhiều ký số như sau: 123456789:

- 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen.
- 5 ký số tiếp theo (34567) sẽ được đọc vào cho bienthuc.
- bienchar sẽ có giá trị là ký số tiếp theo '8'.

```
scanf("%s",chuoi1);
hoặc scanf("%s",chuoi2) ;
```

Nếu ta nhập chuỗi như sau: *Nguyen Van Linh* ↵ thì giá trị của biến chuoi1 hay chuoi2 chỉ là *Nguyen* .

```
scanf("%s%s",chuoi1, chuoi2);
```

Nếu ta nhập chuỗi như sau: *Duong Van Hieu* ↵ thì giá trị của biến chuoi1 là *Duong* và giá trị của biến chuoi2 là *Van.*

Vì sao như vậy? C sẽ đọc từ đầu đến khi gặp khoảng trắng và gán giá trị cho biến đầu tiên, phần còn lại sau khoảng trắng là giá trị của các biến tiếp theo.

```
gets(chuoi1);
```

Nếu nhập chuỗi : *Nguyen Van Linh* ↵ thì giá trị của biến chuoi1 là *Nguyen Van Linh*

II.3. Hiển thị giá trị của biểu thức lên màn hình (hàm printf)

*Hàm printf() (nằm trong thư viện **stdio.h**) dùng để hiển thị (in) giá trị của các biểu thức lên màn hình.*

Cú pháp:

printf("Chuỗi định dạng ", Các biểu thức);

Giải thích:

- *Chuỗi định dạng*: dùng để quy định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân... Bảng dưới đây là một số định dạng khi hiển thị các biểu thức số nguyên, số thực, ký tự,... :

Định dạng	**Ý nghĩa**
%d	In số nguyên
%[.số chữ số thập phân] f	In số thực có <số chữ số thập phân> theo quy tắc làm tròn số.
%o	In số nguyên hệ bát phân
%x	In số nguyên hệ thập lục phân
%c	In một ký tự
%s	In chuỗi ký tự
%e hoặc %E hoặc %g hoặc %G	In số nguyên dạng khoa học (nhân 10 mũ x)
Thí dụ	
%d	In ra số nguyên
%4d	In số nguyên tối đa 4 ký số, nếu số cần in nhiều hơn 4 ký số thì in hết
%f	In số thực
%6f	In số thực tối đa 6 ký số (tính luôn dấu chấm), *nếu số cần in nhiều hơn 6 ký số thì in hết*
%.3f	In số thực có 3 số lẻ, nếu số cần in có nhiều hơn 3 số lẻ thì làm tròn.

- *Các biểu thức*: là các biểu thức mà chúng ta cần hiển thị giá trị của nó lên màn hình, mỗi biểu thức phân cách nhau bởi dấu phẩy (**,**).

Thí dụ:

```
include<stdio.h>
main(){
   int    bien_nguyen=1234, i=65;
   float    bien_thuc=123.456703;
   printf("Gia tri nguyen cua bien nguyen
                =%d\n",bien_nguyen);
```

```
    printf("Gia tri thuc cua bien thuc
                =%f\n",bien_thuc);
    printf("Truoc khi lam tron=%f \n
                Sau khi lam tron=%.2f",
                bien_thuc, bien_thuc);
}
```

Kết quả in ra màn hình như sau:

```
Gia tri nguyen cua bien nguyen =1234
Gia tri thuc cua bien thuc =123.456703
Truoc khi lam tron=123.456703
 Sau khi lam tron=123.46
```

Nếu ta thêm vào dòng sau trong chương trình:

```
printf("\n  Ky tu co ma ASCII %d la %c",i,i);
```

Kết quả ta nhận được thêm:

```
Ky tu co ma ASCII 65 la A_
```

```
printf(" So nguyen la %d \n
        So thuc la %f",i, (float)i );
```

```
So nguyen la 65
So thuc la 65.000000
```

```
printf("\n So thuc la %f \n
   So nguyen la %d",bien_thuc, (int)bien_thuc);
```

```
So thuc la 123.456703
So nguyen la 123_
```

```
printf("\n Viet binh thuong =%f \n
   Viet kieu khoa hoc=%e",bien_thuc, bien_thuc);
```

Kết quả in ra màn hình:

```
Viet binh thuong=123.456703
Viet kieu khoa hoc=1.234567e+02
```

Lưu ý: Đối với các ký tự điều khiển, ta không thể sử dụng cách viết thông thường để hiển thị chúng.

Ký tự điều khiển là các ký tự dùng để điều khiển các thao tác xuất, nhập dữ liệu.

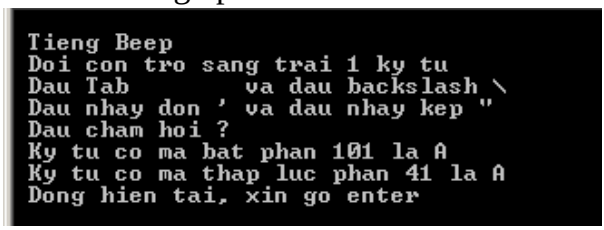
Một số ký tự điều khiển:

Ký tự điều khiển	**Giá trị thập lục phân**	**Ký tự được hiển thị**	**Ý nghĩa**
\a	0x07	BEL	Phát ra tiếng chuông
\b	0x08	BS	Di chuyển con trỏ sang trái 1 ký tự và xóa ký tự bên trái (backspace)
\f	0x0C	FF	Sang trang
\n	0x0A	LF	Xuống dòng
\r	0x0D	CR	Trở về đầu dòng
\t	0x09	HT	Tab theo cột (giống gõ phím Tab)
\\	0x5C	\	Dấu \
\’	0x2C	‘	Dấu nháy đơn (‘)
\”	0x22	“	Dấu nháy kép (“)
\?	0x3F	?	Đấu chấm hỏi (?)
\ddd	ddd	Ký tự có mã ACSII trong hệ bát phân là số ddd	
\xHHH	0xHHH	Ký tự có mã ACSII trong hệ thập lục phân là HHH	

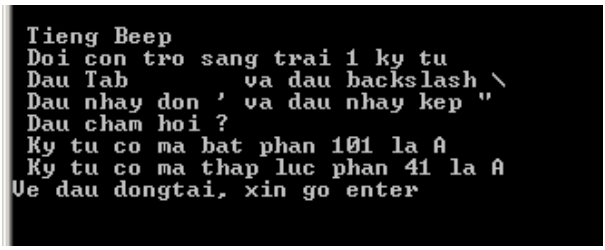
Thí dụ:

```
#include <stdio.h>
#include<conio.h>
main (){
    printf("\n Tieng Beep \a");
    printf("\n Doi con tro sang trai 1 ky tu\b");
    printf("\n Dau Tab \tva dau backslash \\");
    printf("\n Dau nhay don \'
          va dau nhay kep \"");
    printf("\n Dau cham hoi \?");
    printf("\n Ky tu co ma bat phan 101
                la \101");
    printf("\n Ky tu co ma thap luc phan 41
                 la \x041");
    printf("\n Dong hien tai, xin go enter");
    getch();
    printf("\rVe dau dong");
    getch();
}
```

Kết quả trước khi gõ phím Enter:



Kết quả sau khi gõ phím Enter:



III. MỘT THÍ DỤ

Viết chương trình cho phép nhận từ bàn phím 2 số thực biểu diễn cho chiều dài và chiều rộng của 1 hình chữ nhật. Tính chu vi và diện tích của hình chữ nhật đó và hiển thị kết quả lên màn hình.

Với yêu cầu của chương trình này thì đầu vào và đầu ra của chương trình là:

Đầu vào: 2 số thực chiều dài và chiều rộng.

Đầu ra: chu vi và diện tích của hình chữ nhật.

Chương trình sau giải quyết được vấn đề trên:

```
#include<stdio.h>
#include<conio.h>
main()
{
    float Dai, Rong, Chu_Vi, Dien_Tich;
    //1. Nhập chiều dài và chiều rộng
    printf("Chieu dai: ");scanf("%f",&Dai);
    printf("Chieu rong: ");scanf("%f",&Rong);
```

```
    //2. Tính chu vi và diện tích hình chữ nhật
    Chu_Vi = 2*(Dai+Rong);
    Dien_Tich = Dai*Rong;
    //3. Hiển thị kết quả
    printf("Chu vi la %.3f\n
           Dien tich la %.3f",Chu_Vi,Dien_Tich);
    getch();
}
```

IV. BÀI TẬP

1. Viết chương trình in lên màn hình một thiệp mời dự sinh nhật có dạng:

```
**********************************************
                    THIEP MOI
          Thân mời bạn : Nguyễn Mạnh Hùng
          Tới dự lễ sinh nhật của mình
          Vào lúc 19h ngày 12/10/2009
          Tại 05/42 Trần Phú - Cần Thơ
          Rất mong được đón tiếp !
                              Hồ Thu Hương
**********************************************
```

2. Viết chương trình nhập vào bán kính r của một hình tròn. Tính chu vi và diện tích của hình tròn theo công thức :

Chu vi CV = 2*Pi*r
Diện tích S = Pi*r*r
In các kết quả lên màn hình

3. Viết chương trình nhập vào độ dài 3 cạnh a, b, c của một tam giác. Tính chu vi và diện tích của tam giác theo công thức:

Chu vi CV = a+b+c

Diện tích S = sqrt(p*(p-a)*(p-b)*(p-c))

Trong đó: p=CV/2

In các kết quả lên màn hình

4. Viết chương trình tính $\log_a x$ với a, x là các số thực nhập vào từ bàn phím với giả thiết x>0, a>0, a != 1.(dùng $\log_a x = \ln x / \ln a$)

5. Viết chương trình nhập vào tọa độ của hai điểm (x1, y1) và (x2, y2)

a) Tính hệ số góc của đường thẳng đi qua hai điểm đó theo công thức:

Hệ số góc = (y2 - y1) /(x2 - x1)

b) Tính khoảng cách giữa hai điểm theo công thức:

Khoảng cách = $\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

6. Viết chương trình nhập vào một ký tự:

a) In ra mã Ascii của ký tự đó.

b) In ra ký tự kế tiếp của nó.

7. Viết chương trình nhập vào các giá trị điện trở R1, R2, R3 của một mạch điện :

Tính tổng trở theo công thức: $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$

8. Viết chương trình nhập vào điểm ba môn Toán, Lý, Hóa của một học sinh. In ra điểm trung bình của học sinh đó với hai số lẻ thập phân.

9. Viết chương trình nhập vào ngày, tháng, năm. In ra ngày tháng năm theo dạng dd/mm/yy. (dd: ngày, mm: tháng, yy : năm. Thí dụ: 20/11/99)

10. Viết chương trình đảo ngược một số nguyên dương có đúng 3 chữ số.

Chương 4

CÁC LỆNH CÓ CẤU TRÚC

Chương này trình bày về các câu lệnh có cấu trúc trong C. Nội dung chính của chương này gồm:

- *Khối lệnh trong C.*
- *Cấu trúc rẽ nhánh.*
- *Cấu trúc lựa chọn.*
- *Cấu trúc vòng lặp*
- *.Các câu lệnh "đặc biệt".*

I. KHỐI LỆNH

Một dãy các khai báo cùng với các câu lệnh nằm trong cặp dấu ngoặc móc { và } được gọi là một khối lệnh.

Thí dụ 1:

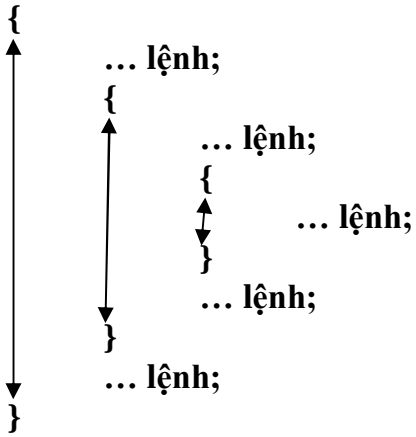
```
{
     char  ten[30];
      printf("\n Nhap vao ten cua ban:");
     scanf("%s", ten);
     printf("\n Chao Ban  %s",ten);
}
```

Thí dụ 2:

```
#include <stdio.h>
#include<conio.h>
main ()
{  /*đây là đầu khối*/
     char ten[50];
     printf("Xin cho biet ten cua ban !");
     scanf("%s",ten);
     getch();
} /*đây là cuối khối*/
```

Một khối lệnh có thể chứa bên trong nó nhiều khối lệnh khác gọi là khối lệnh lồng nhau. Sự lồng nhau của các khối lệnh là không hạn chế.

Minh họa:



Lưu ý về phạm vi tác động của biến trong khối lệnh lồng nhau:

- Trong các khối lệnh khác nhau hay các khối lệnh lồng nhau có thể khai báo các biến cùng tên.

Thí dụ 1:

```
{
        … lệnh;
        {
                int a,b;        /*biến a, b trong khối lệnh thứ nhất*/
                … lệnh;
        }
        …lệnh;
        {
                int a,b;        /*biến a,b trong khối lệnh thứ hai*/
                … lệnh;
        }
}
```

Thí dụ 2:

```
{
        int a, b;                   /*biến a,b trong khối lệnh "bên ngoài"*/
        … lệnh;
        {
                int a,b;            /*biến a,b bên trong khối lệnh con*/
        }
}
```

- Nếu một biến được khai báo bên ngoài khối lệnh và không trùng tên với biến bên trong khối lệnh thì nó cũng được sử dụng bên trong khối lệnh.

- Một khối lệnh con có thể sử dụng các biến bên ngoài, các lệnh bên ngoài không thể sử dụng các biến bên trong khối lệnh con.

Thí dụ:

```
{
        int    a, b, c;
        …lệnh;
        {
                int  c, d;
                …lệnh;
        }
}
```

II. CẤU TRÚC RẼ NHÁNH

Cấu trúc rẽ nhánh là một cấu trúc được dùng rất phổ biến trong các ngôn ngữ lập trình nói chung. Cấu trúc này thể hiện suy nghĩ dạng ***nếu … thì*** *của con người. Cấu trúc rẽ nhánh có hai dạng: dạng không đầy đủ và dạng đầy đủ.*

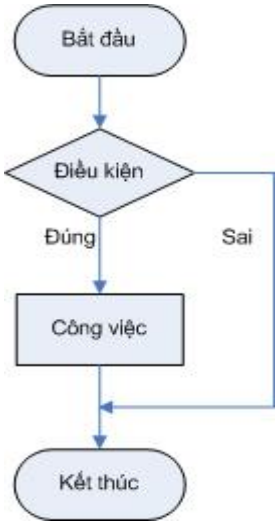
II.1. Dạng không đầy đủ

Cú pháp:

if (<*Biểu thức điều kiện*>**)**

<Công việc>

Lưu đồ cú pháp:



Giải thích:

<Công việc> được thể hiện bằng 1 câu lệnh hay 1 khối lệnh.

Đầu tiên *Điều kiện* được kiểm tra.

Nếu điều kiện đúng (!= 0) thì thực hiện câu lệnh hoặc khối lệnh (*Công việc*) liền sau điều kiện.

Nếu điều kiện sai thì bỏ qua lệnh hoặc khối lệnh liền sau điều kiện (những lệnh và khối lệnh sau đó vẫn được thực hiện bình thường vì nó không phụ thuộc vào điều kiện sau if).

Thí dụ 1: Nhập vào một số thực a từ bàn phím. In ra màn hình kết quả nghịch đảo của a khi a $\neq 0$.

```
#include <stdio.h>
#include <conio.h>
main (){
   float a;
   printf("Nhap a = "); scanf("%f",&a);
   if (a !=0 )
     printf("Nghich dao cua %f la %f",a,1/a);
   getch();
}
```

Giải thích:

- Nếu giá trị nhập vào a $\neq 0$ thì câu lệnh `printf("Nghich dao cua %f la %f",a,1/a)`được thực hiện, ngược lại câu lệnh này không được thực hiện.

- Lệnh getch() luôn luôn được thực hiện vì nó không phải là "lệnh liền sau" điều kiện if.

Thí dụ 2: Nhập vào giá trị của 2 số a và b từ bàn phím, nếu a lớn hơn b thì in ra thông báo "Gia trị của a lớn hơn giá trị của b", sau đó hiển thị giá trị cụ thể của 2 số lên màn hình.

```
#include <stdio.h>
#include<conio.h>
main (){
   int a,b;
   printf("Nhap vao gia tri cua 2 so a, b!");
   scanf("%d%d",&a,&b);
   if (a>b){
      printf("\n Gia tri cua a lon hon
            gia tri cua b");
      printf("\n a=%d,  b=%d",a,b);
   }
   getch();
}
```

Giải thích:

Nếu ta nhập vào giá trị của a lớn hơn giá trị của b thì khối lệnh:

```
   {
          printf("\n Gia tri cua a lon hon
                 gia tri cua b");
          printf("\n a=%d,  b=%d",a,b);
   }
```

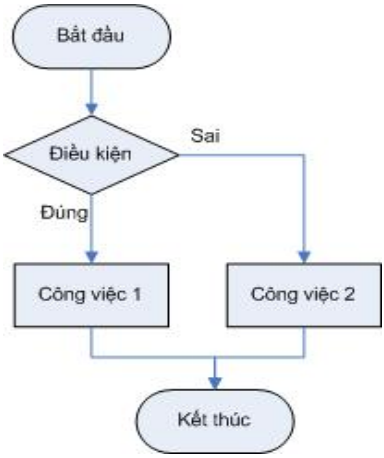
sẽ được thực hiện, ngược lại khối lệnh này không được thực hiện.

II.2. Dạng đầy đủ

Cú pháp:

if (<*Biểu thức điều kiện*>**)**
    <Công việc 1>
else
    <Công việc 2>

Lưu đồ cú pháp:



Giải thích:
Công việc 1, công việc 2 được thể hiện là 1 câu lệnh hay 1 khối lệnh.
Đầu tiên *Biểu thức điều kiện* được kiểm tra trước.
Nếu điều kiện đúng thì thực hiện *công việc 1*.
Nếu điều kiện sai thì thực hiện *công việc 2*.
Các lệnh phía sau *công việc 2* không phụ thuộc vào điều kiện.

Thí dụ 1: Viết chương trình nhập vào một số thực a từ bàn phím. In ra màn hình kết quả nghịch đảo của a khi a $\neq 0$, khi a =0 in ra thông báo "Khong the tim duoc nghich dao cua a"

```
#include <stdio.h>
#include <conio.h>
main (){
   float a;
   printf("Nhap a = "); scanf("%f",&a);
   if (a !=0 )
      printf("Nghich dao cua %f la %f",a,1/a);
   else
      printf("Khong the tim duoc nghich dao
            cua a");
   getch();
}
```

Giải thích:

- Nếu ta nhập vào giá trị a $\neq 0$ thì câu lệnh `printf("Nghich dao cua %f la %f",a,1/a)`được thực hiện, ngược lại câu lệnh `printf("Khong the tim duoc nghich dao cua a")` được thực hiện.

- Lệnh getch() luôn luôn được thực hiện.

Thí dụ 2: Viết chương trình cho phép nhập vào giá trị của 2 số a và b từ bàn phím. nếu a lớn hơn b thì in ra thông báo "a lon hon b" và hiển thị giá trị của 2 số lên màn hình, ngược lại thì in ra màn hình câu thông báo "a nho hon hoặc bang b" và hiển thị giá trị của 2 số lên màn hình.

```
#include <stdio.h>
#include<conio.h>
main (){
   int a, b;
   printf("Nhap vao gia tri cua 2 so a va b !");
   scanf("%d%d",&a,&b);
   if (a>b){
      printf("\n a lon hon b”);
      printf("\n a=%d  b=%d ",a,b);
   }
   else{
      printf("\n a nho hon hoac bang b");
      printf("\n a=%d  b=%d",a,b);
   }
   printf("\n Thuc hien xong lenh if");
   getch();
}
```

Giải thích:

- Nếu ta nhập vào 40 30 ↵ thì kết quả hiển thị trên màn hình là

```
a lon hon b
a=40  b=30
Thuc hien xong lenh if
```

- Còn nếu ta nhập 40 50 ↵ thì kết quả hiển thị trên màn hình là

```
a nho hon hoac bang b
a=40 b=50
Thuc hien xong lenh if
```

Thí dụ 3: Viết chương trình nhập vào một số nguyên dương là tháng trong năm . Hiển thị và số ngày của tháng đó lên màn hình.

Gợi ý: - Tháng có 31 ngày: 1, 3, 5, 7, 8, 10, 12

- Tháng có 30 ngày: 4, 6, 9, 10

- Tháng có 28 hoặc 29 ngày : 2

```
#include <stdio.h>
#include<conio.h>
main (){
   int thg;
   printf("Nhap vao thang trong nam !");
   scanf("%d",&thg);
   if(thg==1||thg==3||thg==5||thg==7||thg==8
           ||thg==10||thg==12)
     printf("\n Thang %d co 31 ngay ",thg);
   else if (thg==4||thg==6||thg==9||thg==11)
     printf("\n Thang %d co 30 ngay",thg);
   else if (thg==2)
     printf("\n Thang %d co 28 hoac
          29 ngay",thg);
   else printf("Khong co thang %d",thg);

   printf("\n Thuc hien xong lenh if");
   getch();
}
```

Giải thích:

- Nếu ta nhập vào một trong các số 1, 3, 5, 7, 8, 10, 12 thì kết quả xuất hiện trên màn hình sẽ là:

Thang <số> co 31 ngay
Thuc hien xong lenh if

- Nếu chúng ta nhập vào một trong các số 4, 6, 9, 11 thì kết quả xuất hiện trên màn hình sẽ là

Thang <số> co 30 ngay
Thuc hien xong lenh if

- Nếu ta nhập vào số 2 thì kết quả xuất hiện trên màn hình sẽ là

```
Thang  2 co 28 hoac 29 ngay
Thuc hien xong lenh if
```

- Nếu ta nhập vào số nhỏ hơn 0 hoặc lớn hơn 12 thì kết quả xuất hiện trên màn hình sẽ là

```
Khong co thang <số>
Thuc hien xong lenh if
```

Trong đó <số> là con số mà chúng ta đã nhập vào.

Lưu ý:

- Ta có thể sử dụng các câu lệnh if…else lồng nhau. Trong trường hợp if…else lồng nhau thì *else sẽ kết hợp với if gần nhất chưa có else*.

- Trong trường hợp câu lệnh if "bên trong" không có else thì phải viết nó trong cặp dấu {} (coi như là khối lệnh) để tránh sự kết hợp else if sai.

Thí dụ 1:

```
if ( so1>0)
      if (so2 > so3)
            a=so2;
      else   /*else của if (so2>so3) */
            a=so3;
```

Thí dụ 2:

```
if (so1>0)
{
      if (so2>so3)    /*lệnh if này không
có else*/
      a=so2;
}
else  /*else của if (so1>0)*/
a=so3;
```

III. CẤU TRÚC LỰA CHỌN

Cấu trúc lựa chọn cho phép lựa chọn một trong nhiều trường hợp. Trong C, đó là câu lệnh switch.

Cú pháp:

```
switch (<Biến>)
{
        case   giá trị 1:
                        Khối lệnh thực hiện công việc 1;
                        break;
                        …
        case    giá trị n:
                        Khối lệnh thực hiện công việc n;
                        break;
        [default        :
                        Khối lệnh thực hiện công việc mặc định;

}
```

Giải thích:

- Trước tiên *Biến* được ước lượng giá trị.
- Nếu giá trị của biểu thức bằng *giá trị 1* thì thực hiện công việc 1 rồi thoát.
- Nếu giá trị của biểu thức khác *giá trị 1* thì so sánh với giá trị 2, nếu bằng giá trị 2 thì thực hiện công việc 2 rồi thoát.
- Cứ như thế, so sánh tới giá trị n.
- Nếu tất cả các phép so sánh trên đều sai thì thực hiện công việc mặc định của trường hợp *default*.

Lưu ý:

- Biểu thức trong switch() phải có kết quả là giá trị kiểu số nguyên (int, char, long, …).
- Các giá trị sau case cũng phải là kiểu số nguyên.
- Không bắt buộc phải có default.

Thí dụ 1: Nhập vào một số nguyên, chia số nguyên này cho 2 lấy phần dư. Kiểm tra nếu phần dư bằng 0 thì in ra thông báo "số chẵn", nếu số dư bằng 1 thì in thông báo "số lẻ".

```
#include <stdio.h>
#include<conio.h>
main (){
   int songuyen, phandu;
   printf("\n Nhap vao so nguyen ");
   scanf("%d",&songuyen);
   phandu=(songuyen % 2);
   switch(phandu){
        case 0:
           printf("%d la so chan ",songuyen);
           break;
        case 1:
           printf("%d la so le ",songuyen);
           break;
     }
   getch();
}
```

Thí dụ 2: Nhập vào 2 số thực và 1 phép toán.

- Nếu phép toán là '+', '-', '*' thì in ra kết qua là tổng, hiệu, tích của 2 số.

- Nếu phép toán là '/' thì kiểm tra xem số thứ 2 có khác không hay không? Nếu khác không thì in ra thương của chúng, ngược lại thì in ra thông báo "khong chia cho 0".

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

main (){
   float a,b;
   char pt;

   printf("Nhap a = ");scanf("%f",&a);
   printf("Nhap b = ");scanf("%f",&b);
   fflush(stdin);
   printf("Phep toan = ");scanf("%c",&pt);

   switch(pt){
```

```
            case '+':
                  printf("%f + %f = %f",a,b,a+b);
                  break;

            case '-':
                  printf("%f - %f = %f",a,b,a-b);
                  break;

            case '*':
                  printf("%f * %f = %f",a,b,a*b);
                  break;

            case '/':
                  if (b==0) printf("Khong chia dc");
                  else printf("%f / %f = %f",
                              a,b,a/b);
                  break;

            default:
                  printf("Khong co phep toan %c",pt);
      }

      getch();
}
```

Trong cấu trúc lựa chọn **switch**…**case**, từ khóa **break** là không bắt buộc. Tuy nhiên, nếu không có từ khóa **break** ở cuối mỗi **case**; khi biến trong phần **switch** bằng với 1 giá trị nào đó, phần công việc sau **case** tương ứng được thực hiện, sau đó các phần công việc sau **case** kế tiếp cũng được thực hiện.

Thí dụ 3: Yêu cầu người thực hiện chương trình nhập vào một số nguyên dương là tháng trong năm và in ra số ngày của tháng đó.

- Tháng có 31 ngày: 1, 3, 5, 7, 8, 10, 12

- Tháng có 30 ngày: 4, 6, 9, 10

- Tháng có 28 hoặc 29 ngày : 2

- Nếu nhập vào số <1 hoặc >12 thì in ra câu thông báo “không có tháng này “.

```
#include <stdio.h>
#include<conio.h>
main (){
   int thang;
   printf("\n Nhap vao thang trong nam ");
   scanf("%d",&thang);
   switch(thang){
      case 1:
      case 3:
      case 5:
      case 7:
      case 8:
      case 10:
      case 12:
         printf("\n Thang %d co 31 ngay ",thang);
         break;

      case 4:
      case 6:
      case 9:
      case 11:
         printf("\n Thang %d co 30 ngay ",thang);
         break;

      case 2:
        printf ("\ Thang 2 co 28 hoac 29 ngay");
        break;

      default:
        printf("\n Khong co thang %d", thang);
        break;
      }
   getch();
}
```

IV. VÒNG LẶP

Cấu trúc vòng lặp cho phép lặp lại nhiều lần 1 công việc (được thể hiện bằng 1 câu lệnh hay 1 khối lệnh) nào đó cho đến khi thỏa mãn 1 điều kiện cụ thể.

IV.1. Vòng lặp for

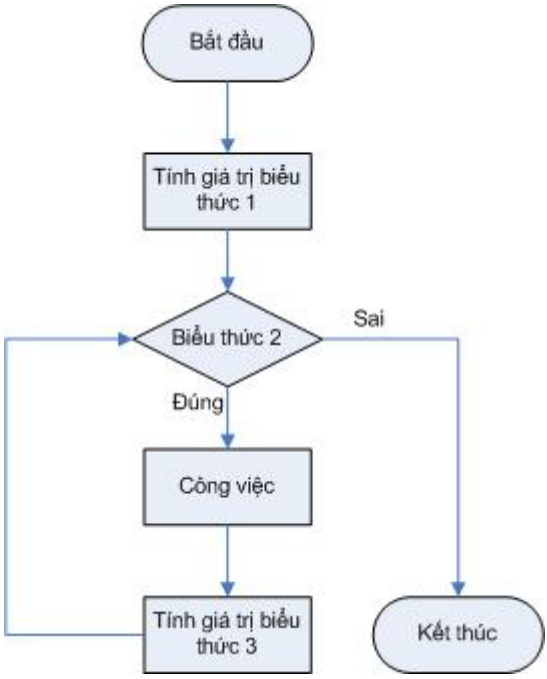
Vòng lặp này cho phép lặp lại công việc trong khi điều kiện còn đúng.

Cú pháp:

for (Biểu thức 1; biểu thức 2; biểu thức 3)

<Công việc>

Lưu đồ:



Giải thích:

<Công việc>: được thể hiện là 1 câu lệnh hay 1 khối lệnh. Thứ tự thực hiện của câu lệnh for như sau:

Bước 1: Tính giá trị của biểu thức 1.

Bước 2: Tính giá trị của biểu thức 2.

- Nếu giá trị của biểu thức 2 là sai (=0): *thoát khỏi câu lệnh for*.

- Nếu giá trị của biểu thức 2 là đúng (!=0): <Công việc> được thực hiện.

Bước 3: Tính giá trị của biểu thức 3 và quay lại bước 2.

Một số lưu ý khi sử dụng câu lệnh for:

- Khi biểu thức 2 vắng mặt thì nó được coi là luôn luôn đúng

- Biểu thức 1: thông thường là một phép gán để khởi tạo giá trị ban đầu cho biến điều kiện.

- Biểu thức 2: là một biểu thức kiểm tra điều kiện đúng sai để tiếp tục hay dừng vòng lặp.

- Biểu thức 3: thông thường là một phép gán để thay đổi giá trị của biến điều kiện.

- Trong mỗi biểu thức có thể có nhiều biểu thức con. Các biểu thức con được phân biệt bởi dấu phẩy.

Thí dụ 1: Viết chương trình in dãy số nguyên từ 1 đến 10.

```
#include <stdio.h>
#include<conio.h>
main (){
   int i;
   printf("\n Day so tu 1 den 10 :");
   for (i=1; i<=10; i++)
     printf("%d ",i);
   getch();
}
```

Kết quả chương trình như sau:

```
Day so tu 1 den 10 :1 2 3 4 5 6 7 8 9 10
```

Thí dụ 2: Viết chương trình nhập vào một số nguyên n. Tính tổng của các số nguyên từ 1 đến n.

```
#include <stdio.h>
#include<conio.h>
main (){
   unsigned int n,i,tong;
   printf("Nhap vao so nguyen duong n:");
   scanf("%u",&n);

   tong=0;
   for (i=1; i<=n; i++)
      tong+=i;
   printf("\n Tong tu 1 den %u =%u ",n,tong);
   getch();
}
```

Nếu chúng ta nhập vào số 9 thì kết quả như sau:

```
Nhap vao so nguyen duong n:9

Tong tu 1 den 9 =45 _
```

Thí dụ 3: Viết chương trình in ra trên màn hình một ma trận có n dòng m cột như sau:

```
1    2    3    4    5    6    7
2    3    4    5    6    7    8
3    4    5    6    7    8    9
…
```

```
#include <stdio.h>
#include<conio.h>
main (){
   unsigned int dong, cot, n, m;
   printf("\n Nhap vao so dong va so cot :");
   scanf("%u%u",&n,&m);
   for (dong=0;dong<n;dong++){
       printf("\n");
       for (cot=1;cot<=m;cot++)
            printf("%u\t",dong+cot);
   }
   getch();
}
```

Kết quả khi nhập 3 dòng 6 cột như sau

```
Nhap vao so dong va so cot :3 6

1       2       3       4       5       6
2       3       4       5       6       7
3       4       5       6       7       8
```

III.2. Vòng lặp while

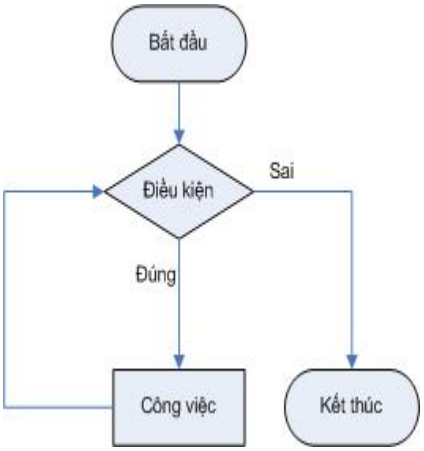
Vòng lặp while giống như vòng lặp for, dùng để lặp lại một công việc nào đó trong khi điều kiện còn đúng.

Cú pháp:

while (*Biểu thức điều kiện***)**

<Công việc>

Lưu đồ:



Giải thích:

- *Công việc*: được thể hiện bằng 1 câu lệnh hay 1 khối lệnh.
- Trước tiên *điều kiện* được kiểm tra.
- Nếu điều kiện sai (=0) thì thoát khỏi lệnh while.
- Nếu điều kiện đúng (!=0) thì thực hiện công việc rồi quay lại kiểm tra điều kiện tiếp.

Lưu ý:

- Lệnh **while** gồm có biểu thức điều kiện và thân vòng lặp (khối lệnh thực hiện công việc)

- Vòng lặp dừng lại khi điều kiện sai.

- Khối lệnh thực hiện công việc có thể rỗng, có thể làm thay đổi điều kiện.

Thí dụ 1: Viết chương trình in dãy số nguyên từ 1 đến 10.

```
#include <stdio.h>
#include<conio.h>
main (){
   int i;
   printf("\n Day so tu 1 den 10 :");
   i=1;

   while (i<=10){
      printf("%d ",i);
      i++;
   }
   getch();
}
```

Kết quả chương trình như sau:

```
Day so tu 1 den 10 :1 2 3 4 5 6 7 8 9 10
```

Thí dụ 2: Viết chương trình nhập vào một số nguyên n. Tính tổng của các số nguyên từ 1 đến n.

```
#include <stdio.h>
#include<conio.h>
main (){
   unsigned int n,i,tong;
   printf("\n Nhap vao so nguyen duong n:");
   scanf("%u",&n);
   tong=0;
   i=1;
   while (i<=n){
      tong+=i;
      i++;
   }
   printf("\n Tong tu 1 den %u =%u ",n,tong);
   getch();
}
```

Nếu chúng ta nhập vào số 9 thì kết quả như sau:

```
Nhap vao so nguyen duong n:9

Tong tu 1 den 9 =45 _
```

Thí dụ 3: Viết chương trình in ra trên màn hình một ma trận có n dòng m cột như sau:

```
1    2    3    4    5    6    7
2    3    4    5    6    7    8
3    4    5    6    7    8    9
…
```

```
#include <stdio.h>
#include<conio.h>
main (){
   unsigned int dong, cot, n, m;
   printf("\n Nhap vao so dong va so cot :");
   scanf("%u%u",&n,&m);
   dong=0;
   while (dong<n){
       printf("\n");
       cot=1;
       while (cot<=m){
            printf("%u\t",dong+cot);
            cot++;
       }
       dong++;
   }
   getch();
}
```

Kết quả khi nhập 3 dòng 6 cột như sau:

```
Nhap vao so dong va so cot :3 6

1       2       3       4       5       6
2       3       4       5       6       7
3       4       5       6       7       8
```

IV.3. Vòng lặp do… while

Vòng lặp do … while giống như vòng lặp for, while, dùng để lặp lại một công việc nào đó trongkhi điều kiện còn đúng.

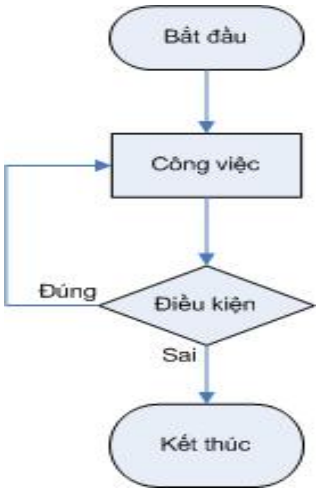
Cú pháp:

do

<Công việc>

while (<*Biểu thức điều kiện*>**)**

Lưu đồ:



Giải thích:

- <Công việc>: được thể hiện bằng 1 câu lệnh hay 1 khối lệnh.
- Trước tiên **công việc** được thực hiện trước, sau đó mới kiểm tra *điều kiện* mới được kiểm tra.
- Nếu điều kiện sai thì thoát khỏi lệnh **do …while**.
- Nếu điều kiện còn đúng thì thực hiện công việc rồi quay lại kiểm tra điều kiện tiếp...

Lưu ý:

- Lệnh do…while thực hiện công việc ít nhất 1 lần.

- Vòng lặp dừng lại khi điều kiện sai.

- Khối lệnh thực hiện công việc có thể rỗng, có thể làm thay đổi điều kiện.

Thí dụ 1: Viết chương trình in dãy số nguyên từ 1 đến 10.

```
#include <stdio.h>
#include<conio.h>
main (){
   int i;
   printf("\n Day so tu 1 den 10 :");
   i=1;
   do{
      printf("%d ",i);
      i++;
   }while (i<=10);
   getch();
}
```

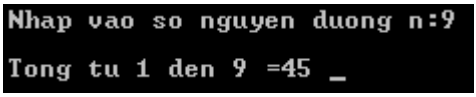
Kết quả chương trình như sau:

```
Day so tu 1 den 10 :1 2 3 4 5 6 7 8 9 10
```

Thí dụ 2: Viết chương trình nhập vào một số nguyên n. Tính tổng của các số nguyên từ 1 đến n.

```
#include <stdio.h>
#include<conio.h>
main (){
   unsigned int n,i,tong;
   printf("\n Nhap vao so nguyen duong n:");
   scanf("%u",&n);
   tong=0;
   i=1;
   do{
      tong+=i;
      i++;
   } while (i<=n);
   printf("\n Tong tu 1 den %u =%u ",n,tong);
   getch();
}
```

Nếu chúng ta nhập vào số 9 thì kết quả như sau:



Thí dụ 3: Viết chương trình in ra trên màn hình một ma trận có n dòng m cột như sau (n, m>=1):

```
1   2   3   4   5   6   7
2   3   4   5   6   7   8
3   4   5   6   7   8   9
…
```

```
#include <stdio.h>
#include<conio.h>
main (){
   unsigned int dong, cot, n, m;
   printf("\n Nhap vao so dong va so cot :");
   scanf("%u%u",&n,&m);
   dong=0;
   do{
      printf("\n");
      cot=1;

      do{
```

```
            printf("%u\t",dong+cot);
            cot++;
         } while (cot<=m);
         dong++;
      } while (dong<n);
      getch();
}
```

Kết quả khi nhập 3 dòng 6 cột như sau

```
Nhap vao so dong va so cot :3 6

1       2       3       4       5       6
2       3       4       5       6       7
3       4       5       6       7       8
```

IV.4. So sánh các vòng lặp

Vòng lặp for, while:

- Kiểm tra điều kiện trước thực hiện công việc sau nên đoạn lệnh thực hiện công việc có thể không được thực hiện .

- Vòng lặp kết thúc khi điều kiện sai.

Vòng lặp do…while:

- Thực hiện công việc trước kiểm tra điều kiện sau nên đoạn lệnh thực hiện công việc được thực hiện ít nhất 1 lần.

- Vòng lặp kết thúc khi điều kiện sai.

V. CÁC CÂU LỆNH ĐẶC BIỆT

V.1. Lệnh break

Cú pháp: break;

Dùng để thoát khỏi vòng lặp. Khi gặp câu lệnh này trong vòng lặp, chương trình sẽ thoát ra khỏi vòng lặp và chỉ đến câu lệnh liền sau nó. Nếu nhiều vòng lặp thì break sẽ thoát ra khỏi vòng lặp gần nhất.

Bên cạnh đó, break còn được dùng trong cấu trúc lựa chọn switch.

Thí dụ: Viết chương trình nhập một số nguyên dương n từ bàn phím. Kiểm tra n có là số nguyên tố hay không?

```
#include<stdio.h>
#include<conio.h>
main(){
    unsigned int n,i;
    printf("Nhap n= ");scanf("%u",&n);
    for(i=2;i<=n-1;i++)
       if (n%i==0) break;

    if(i==n) printf("%u la so nguyen to",n);
    else printf("%u khong la so nguyen to",n);

    getch();
}
```

IV.2. Lệnh continue

Cú pháp: **continue;**

- Khi gặp lệnh này trong các vòng lặp, chương trình sẽ bỏ qua phần còn lại trong vòng lặp và tiếp tục thực hiện lần lặp tiếp theo.

- Đối với lệnh for, *biểu thức 3* sẽ được tính trị và quay lại bước 2.

- Đối với lệnh while, do while; *biểu thức điều kiện* sẽ được tính và xét xem có thể tiếp tục thực hiện <*Công việc*> nữa hay không? (dựa vào kết quả của *biểu thức điều kiện*).

Thí dụ: Nhập vào một số nguyên dương n từ bàn phím. Hiển thị các số lẻ từ 1 đến n.

```
#include<stdio.h>
#include<conio.h>
int main(){
    unsigned int n,i;
```

```
        printf("Nhap n= ");scanf("%u",&n);
        for(i=1;i<=n;i++){
           if(i%2==0)  continue;
           printf("%d ",i);
        }
        getch();
    }
```

VI. BÀI TẬP

1. Viết chương trình nhập 3 số từ bàn phím, tìm số lớn nhất trong 3 số đó, in kết quả lên màn hình.

2. Viết chương trình tính chu vi, diện tích của tam giác với yêu cầu sau khi nhập 3 số a, b, c phải kiểm tra lại xem a, b, c có tạo thành một tam giác không? Nếu có thì tính chu vi và diện tích. Nếu không thì in ra câu " Không tạo thành tam giác".

3. Viết chương trình giải phương trình bậc nhất ax+b=0 với a, b nhập từ bàn phím.

4. Viết chương trình giải phương trình bậc hai $ax^2+bx + c = 0$ với a, b, c nhập từ bàn phím.

5. Viết chương trình nhập từ bàn phím 2 số a, b và một ký tự ch.

Nếu: ch là "+" thì thực hiện phép tính a + b và in kết quả lên màn hình.

ch là "–" thì thực hiện phép tính a - b và in kết quả lên màn hình.

ch là "*" thì thực hiện phép tính a * b và in kết quả lên màn hình.

ch là "/" thì thực hiện phép tính a / b và in kết quả lên màn hình.

6. Viết chương trình nhập vào 2 số là tháng và năm của một năm. Xét xem tháng đó có bao nhiêu ngày? Biết rằng:

Nếu tháng là 4, 6, 9, 11 thì số ngày là 30.

Nếu tháng là 1, 3, 5, 7, 8, 10, 12 thì số ngày là 31.

Nếu tháng là 2 và năm nhuận thì số ngày 29, ngược lại thì số ngày là 28.

7. Viết chương trình tính tiền điện gồm các khoản sau:

Tiền thuê bao điện kế : 1000 đồng / tháng.

Định mức sử dụng điện cho mỗi hộ là 50 Kw

Phần định mức tính giá 450 đồng /Kwh

Nếu phần vượt định mức <= 50 Kw tính giá phạt cho phần này là 700 đồng/Kwh .

Nếu phần vượt định mức lớn 50 Kw và nhỏ hơn 100Kw tính giá phạt cho phần này là 910 đồng/Kwh

Nếu phần vượt định mức lớn hơn hay bằng 100 Kw tính giá phạt cho phần này là 1200 đồng/Kwh .

Với : chỉ số điện kế cũ và chỉ số điện kế mới nhập vào từ bàn phím. In ra màn hình số tiền trả trong định mức, vượt định mức và tổng của chúng.

8. Kiểm tra một ký tự nhập vào thuộc tập hợp nào trong các tập ký tự sau:

Các ký tự chữ hoa: 'A' ...'Z'

Các ký tự chữ thường: 'a' ... 'z'

Các ký tự chữ số : '0' ... '9'

Các ký tự khác.

9. Hệ thập lục phân dùng 16 ký số bao gồm các ký tự 0 .. 9 và A, B, C, D, E ,F.

Các ký số A, B, C, D, E, F có giá trị tương ứng trong hệ thập phân như sau:

A	10
B	11
C	12
D	13
E	14
F	15

Hãy viết chương trình cho nhập vào ký tự biểu diễn một ký số của hệ thập lục phân và cho biết giá trị thập phân tương

ứng. Trường hợp ký tự nhập vào không thuộc các ký số trên, đưa ra thông báo lỗi :

"Hệ thập lục phân không dùng ký số này"

10. Viết chương trình nhập vào ngày tháng năm của ngày hôm nay, in ra ngày tháng năm của ngày mai.

11. Viết chương trình tính các tổng sau:

a) S=1 + 2 + ... + n

b) S=1/2 + 2/3 + ... +n/(n+1)

c) S= - 1 +2 - 3 +4 - ... + $(-1)^n$n

12. Viết chương trình tính P=2*4*6*...*(2n), n nhập từ bàn phím.

13. Viết chương trình nhập vào một dãy n số, tìm số lớn nhất của dãy và xác định vị trí của số lớn nhất trong dãy.

14. Tính giá trị trung bình của một dãy số thực nhập từ bàn phím, kết thúc dãy nhập khi nhập -1.

15. Fibonacci là một dãy số được định nghĩa như sau:

$$F_n = \begin{cases} 1, & \text{nếu } =0 \text{ hoặc } =1 \\ F_{n-1} + F_{n-2}, & n > 1 \end{cases}$$

Viết chương trình in ra màn hình dãy Fibonacci có n số hạng, n nhập từ bàn phím khi cho chạy chương trình.

16. Viết chương trình đếm số chữ số của một số nguyên n.

17. Viết chương trình in ra số đảo ngược của một số nguyên n, với n nhập từ bàn phím.

18. Tìm số nguyên dương k nhỏ nhất sao cho 2^k > n với n là một số nguyên dương nhập từ bàn phím.

19. Viết chương trình mô phỏng phép chia nguyên DIV 2 số nguyên a và b như sau: để chia nguyên a và b ta tính trị a-b, sau đó lấy hiệu tìm được lại trừ cho b... tiếp tục cho đến khi hiệu của nó nhỏ hơn b. Số lần thực hiện được các phép trừ ở trên sẽ bằng trị của phép chia nguyên.

20. Tìm số nguyên dương N nhỏ nhất sao cho

1+1/2+ ...+1/N > S, với S nhập từ bàn phím.

21. Viết chương trình tìm UCLN và BCNN của hai số a và b theo thuật toán sau (Ký hiệu UCLN của a, b là (a,b) còn BCNN là [a,b])

- Nếu a chia hết cho b thì (a,b) = b
- Nếu a = b*q + r thì (a,b) = (b,r)
- [a,b] = a*b/(b,r)

22. Viết chương trình nhập vào một số nguyên dương n, in ra màn hình các số nguyên tố p <= n. Số nguyên p gọi là số nguyên tố nếu p chỉ chia hết cho một và chia hết cho bản thân nó.

23. Viết chương trình tính gần đúng căn bậc hai của một số dương a theo phương pháp Newton:

Trước hết cho $x_0=(1 + a)/2$ sau đó là công thức truy hồi:

$$x_{n+1}=(x_n + a/x_n)/2$$

Nếu: $\left|\frac{x_{n+1} - x_n}{x_n}\right| < e$ thì căn bậc hai của a bằng

Trong đó e là một hằng số cho trước làm độ chính xác.

24. Viết chương trình tính gần đúng căn bậc n của một số dương a theo phương pháp Newton :

Trước hết cho $x_0= a/n$ sau đó là công thức truy hồi:

$$x_{k+1} = \left|\frac{(n-1)\, x_k^{\,n} + a}{n x_k^{\,n-1}}\right|$$

Nếu $|a- x_n^n| < e$ thì x_n là căn bậc n của a. Trong đó e là một hằng số cho trước làm độ chính xác. Nếu a < 0 và n chẵn thì không tồn tại căn.

Chương 5

CHƯƠNG TRÌNH CON (HÀM)

Chương này trình bày về chương trình con (hàm) trong C. Các nội dung chính của chương này như sau:

- *Khái niệm về hàm (function) trong C.*
- *Cách xây dựng và cách sử dụng hàm trong C.*

I. KHÁI NIỆM HÀM

Trong những chương trình lớn, có thể có những đoạn chương trình viết lặp đi lặp lại nhiều lần, để tránh rườm rà và mất thời gian khi viết chương trình; người ta thường phân chia chương trình thành nhiều module, mỗi module giải quyết một công việc nào đó. Các module như vậy gọi là các chương trình con.

Một tiện lợi khác của việc sử dụng chương trình con là ta có thể dễ dàng kiểm tra xác định tính đúng đắn của nó trước khi ráp nối vào chương trình chính và do đó việc xác định sai sót để tiến hành hiệu đính trong chương trình chính sẽ thuận lợi hơn.

Trong C, chương trình con được gọi là hàm. Hàm trong C có thể trả về kết quả thông qua tên hàm hay có thể không trả về kết quả.

Hàm có hai loại: hàm chuẩn và hàm tự định nghĩa (hàm người dùng). Trong chương này, ta chú trọng đến cách định nghĩa hàm và cách sử dụng các hàm đã được định nghĩa.

Một hàm khi được định nghĩa thì có thể sử dụng bất cứ đâu trong chương trình. Để ý rằng trong C, một chương trình bắt đầu thực thi bằng hàm main.

Thí dụ 1: Ta có hàm max để tìm số lớn nhất giữa 2 số nguyên a, b như sau:

```
int max(int a, int b){
    return (a>b) ? a:b;
}
```

Thí dụ 2: Ta có chương trình chính (hàm main) dùng để nhập vào 2 số nguyên a,b và in ra màn hình số lớn trong 2 số

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b){
    return (a>b) ? a:b;
}

main(){
   int a, b, c;
   printf("\n Nhap vao 3 so a, b,c  ");
   scanf("%d%d%d",&a,&b,&c);
   printf("\n So lon la %d",max(a, max(b,c)));
   getch();
}
```

I.1. Hàm chuẩn

Hàm chuẩn là những hàm đã được định nghĩa sẵn trong một thư viện nào đó. Để sử dụng các hàm thư viện thì các thư viện định nghĩa chúng phải được tham chiếu đến nhờ chỉ thị tiền xử lý *#include <tên thư viện.h>*

Một số thư viện thường dùng:

1. stdio.h : Thư viện chứa các hàm vào/ra chuẩn (**st**an**d**ard **i**nput/**o**utput). Gồm các hàm **printf(), scanf(), getc(), putc(), gets(), puts(), fflush(), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()…**

2. conio.h : Thư viện chứa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm **getch(), getche(), getpass(), cgets(), cputs(), putch(), ,…**

3. math.h: Thư viện chứa các hàm tính toán gồm các hàm **abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…**

4. malloc.h: Thư viện chứa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm **calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …**

5. io.h: Thư viện chứa các hàm vào ra cấp thấp. Gồm các hàm **open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),…**

…

I.2. Hàm người dùng

Hàm người dùng là những hàm do người lập trình tự tạo ra nhằm đáp ứng nhu cầu xử lý của mình.

II. ĐỊNH NGHĨA VÀ SỬ DỤNG HÀM

II.1 Định nghĩa hàm

Cấu trúc của một hàm tự thiết kế:

<*kiểu kết quả*> Tên hàm ([<*kiểu t số*> <*tham số*>]
[,<*kiểu t số*><*tham số*>][…])
{
[Khai báo biến cục bộ và các câu lệnh thực hiện hàm]
[return [<Biểu thức>];]
}

Giải thích:

- *Kiểu kết quả:* là kiểu dữ liệu của kết quả trả về, có thể là: int, byte, char, float, void… Một hàm có thể có hoặc không có kết quả trả về. Trong trường hợp *hàm không có kết quả trả về ta nên sử dụng kiểu kết quả là void.*

- *Kiểu t số*: là kiểu dữ liệu của tham số.

- *Tham số:* là tham số truyền dữ liệu vào cho hàm, một hàm có thể có hoặc không có tham số. ***Tham số này gọi là tham số hình thức*, *khi gọi hàm chúng ta phải truyền cho nó các giá trị thực tế (tham số thực tế).*** Nếu có nhiều tham số, mỗi tham số phân cách nhau dấu phẩy (,).

- Bên trong thân hàm (phần giới hạn bởi cặp dấu {}) là các khai báo cùng các câu lệnh xử lý. Các khai báo bên trong hàm được gọi là các khai báo cục bộ trong hàm và các khai báo này chỉ tồn tại bên trong hàm mà thôi.

- Khi định nghĩa hàm, ta thường sử dụng câu lệnh return để trả về kết quả thông qua tên hàm.

Lệnh return dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.

Cú pháp:

return ; **/*không trả về giá trị*/**

return <biểu thức>;//Trả về giá trị của biểu thức

return (<biểu thức>); //Trả về giá trị của biểu thức

Nếu hàm có kết quả trả về, ta bắt buộc phải sử dụng câu lệnh return để trả về kết quả cho hàm.

Thí dụ 1: Viết hàm tìm số lớn giữa 2 số nguyên a và b

```
int max(int a, int b){
    return (a>b) ? a:b;
}
```

Thí dụ 2: Viết hàm tìm ước chung lớn nhất giữa 2 số nguyên a, b.

Cách tìm: đầu tiên ta giả sử UCLN của hai số là số nhỏ nhất trong hai số đó. Nếu điều đó không đúng thì ta giảm đi một đơn vị và cứ giảm như vậy cho tới khi nào tìm thấy UCLN.

```
usigned int  ucln(unsigned int a,
                          unsigned int b){
   unsgined int u;
   if (a<b)
        u=a;
   else
        u=b;
   while ((a%u !=0) || (b%u!=0))
        u--;
   return u;
}
```

II.2 Sử dụng hàm

Một hàm khi định nghĩa thì chúng vẫn chưa được thực thi trừ khi ta có một lời gọi đến hàm đó.

Cú pháp gọi hàm: <Tên hàm>([Danh sách các tham số])

Thí dụ: Viết chương trình cho phép tìm ước số chung lớn nhất của hai số tự nhiên.

```
#include<stdio.h>
unsigned int  ucln(unsigned  int a,
                   unsigned  int b)
{
   unsigned  int u;
   if (a<b)
        u=a;
   else
        u=b;
   while ((a%u !=0) || (b%u!=0))
        u--;
   return u;
}

main(){
   unsigned int A, B, UC;
   printf("Nhap a,b: ");scanf("%u%u",&A,&B);
   UC = ucln(A,B);
   printf("Uoc chung lon nhat la: %u", UC);
}
```

Lưu ý:

- Khi 1 hàm có giá trị trả về, lời gọi hàm là một biểu thức, không phải là một câu lệnh.

- Khi 1 hàm không có giá trị trả về (void), lời gọi hàm được coi tương đương như 1 câu lệnh.

II.3 Nguyên tắc hoạt động của hàm

Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi. Quá trình diễn ra như sau:

- Nếu hàm có tham số, trước tiên các tham số sẽ được *gán giá trị thực tế tương ứng* (giá trị thực tế chính là giá trị chỉ ra trong lời gọi hàm).

- Chương trình sẽ thực hiện tiếp các câu lệnh trong thân hàm bắt đầu từ lệnh đầu tiên đến câu lệnh cuối cùng.

Thí dụ: Lời gọi hàm ucln(A, B) trong hàm main ở trên:

- A, B chính là các giá trị thực tế (tham số thực tế).
- a, b trong định nghĩa của hàm ucln (dòng `unsigned int ucln (unsigned int a, unsigned int b)`) là tham số hình thức.
- Khi có lời gọi hàm ucln(A, B) thì a sẽ nhận giá trị của A, b sẽ nhận giá trị của B; hàm ucln sẽ thực hiện từ đầu đến cuối hàm với các giá trị a, b đã nhận được.

- Khi gặp lệnh return hoặc dấu } cuối cùng trong thân hàm, chương trình sẽ thoát khỏi hàm để trở về chương trình gọi nó và thực hiện tiếp tục những câu lệnh của chương trình này.

III. TRUYỀN THAM SỐ CHO HÀM

Mặc nhiên, việc truyền tham số cho hàm trong C là truyền theo giá trị; nghĩa là tham số hình thức sẽ chỉ nhận giá trị là giá trị của tham số thực tế tương ứng. Như vậy, về cơ bản tham số hình thức và tham số thực tế là hoàn toàn khác nhau; do đó tham số thực tế không bị thay đổi giá trị sau khi hàm vừa được thực thi xong.

Thí dụ 1: Giả sử ta muốn in ra nhiều dòng, mỗi dòng 50 ký tự nào đó. Để đơn giản ta viết một hàm, nhiệm vụ của hàm này là in ra trên một dòng 50 ký tự nào đó. Hàm này có tên là InKT.

```
#include <stdio.h>
#include <conio.h>
void InKT(char ch){
      int i;
      for(i=1;i<=50;i++)
            printf("%c",ch);
      printf("\n");
}
```

```
main(){
      char c;
      InKT('*'); // In ra 50 dau *
      InKT('+');
      c = 'A';
      InKT(c);
}
```

Lưu ý:

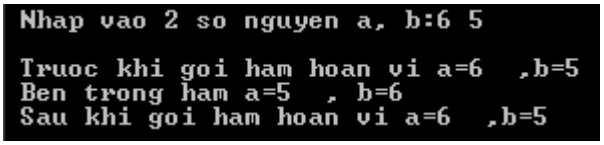
- Trong hàm InKT ở trên, biến ch gọi là tham số hình thức được truyền bằng giá trị (gọi là tham trị của hàm). Các tham trị của hàm coi như là một biến cục bộ trong hàm và chúng được sử dụng như là dữ liệu đầu vào của hàm.

- Khi chương trình con được gọi để thi hành, tham trị được cấp ô nhớ và nhận giá trị là bản sao giá trị của tham số thực. Do đó, mặc dù tham trị cũng là biến, nhưng việc thay đổi giá trị của chúng không có ý nghĩa gì đối với bên ngoài hàm, không ảnh hưởng đến chương trình chính, nghĩa là không làm ảnh hưởng đến tham số thực tương ứng.

Thí dụ 2: Xét chương trình sau đây:

```
#include <stdio.h>
#include <conio.h>
void  hoanvi(int a, int b){
      int t;
      t=a;  //Hoán vị giá trị của 2 biến a, b
      a=b;
      b=t;
      printf("\Ben trong ham a=%d, b=%d",a,b);
}

main(){
      int a, b;
      printf("\nNhap vao 2 so nguyen a, b:");
      scanf("%d%d",&a,&b);
      printf("\nTruoc khi goi ham hoan vi a=%d,
                  b=%d",a,b);
      hoanvi(a,b);
      printf("\nSau khi goi ham hoan vi a=%d  ,
                  b=%d",a,b);
      getch();
}
```

Kết quả thực hiện chương trình:



Tương tự như thí dụ trên, các tham số thực tế (a, b trong hàm main) không thay đổi giá trị sau khi gọi hàm hoanvi.

IV. HÀM ĐỆ QUY

IV.1. Định nghĩa

Một hàm được gọi là đệ quy nếu bên trong thân hàm có lệnh gọi đến chính nó.

Thí dụ: Người ta định nghĩa giai thừa của một số nguyên dương n như sau:

$$n! = \begin{cases} 1, n = 0 \\ n * (n-1)!, n > 0 \end{cases}$$

Với định nghĩa trên thì hàm đệ quy tính n! được viết:

```
long giaithua_dequy(int n){
      if (n==0)
            return 1L;
      else
            return n*giaithua_dequy(n-1);
}

      /*Hàm tính n! không đệ quy*/
long giaithua_khongdequy(int n){
      long kq;
      int i;
      kq=1L;
      for (i=1;i<=n;i++)
            kq=kq*i;
      return kq;
}
```

```
main(){
      int n;
      printf("Nhap so n = ");scanf("%d",&n);
      printf("\nGoi ham de quy: %d!= %ld",
                  n,giaithua_dequy(n));
      printf("\nGoi ham khong de quy: %d!=%ld",
                  n,giaithua_khongdequy(n));
      getch();
}
```

IV.2. Lưu ý khi viết hàm đệ quy

- Hàm đệ quy phải có 2 phần:

  - Phần dừng (hay trường hợp nguyên tố). Trong thí dụ ở trên thì trường hợp n=0 là trường hợp nguyên tố.

  - Phần đệ quy: là phần có gọi lại hàm đang được định nghĩa. Trong thí dụ trên thì phần đệ quy là n>0 thì n! = n * (n-1)!

- Sử dụng hàm đệ quy trong chương trình sẽ làm chương trình dễ đọc, dễ hiểu và vấn đề được nêu bật rõ ràng hơn. Tuy nhiên trong đa số trường hợp thì hàm đệ quy tốn bộ nhớ nhiều hơn và tốc độ thực hiện chương trình chậm hơn không đệ quy.

- Tùy từng bài toán cụ thể mà người lập trình quyết định có nên dùng đệ quy hay không (có những trường hợp không dùng đệ quy thì không giải quyết được bài toán).

V. BÀI TẬP

1. Viết hàm tìm số lớn nhất trong hai số. Áp dụng tìm số lớn nhất trong ba số a, b, c với a, b, c nhập từ bàn phím.

2. Viết hàm tìm UCLN của hai số a và b. Áp dụng: Nhập vào tử và mẫu số của một phân số, kiểm tra xem phân số đó đã tối giản hay chưa.

3. Viết hàm in n ký tự c trên một dòng. Viết chương trình cho nhập 5 số nguyên cho biết số lượng hàng bán được của mặt hàng A ở 5 cửa hàng khác nhau. Dùng hàm trên thể hiện biểu đồ so sánh 5 giá trị đó, mỗi trị dùng một ký tự riêng.

4. Viết một hàm tính tổng các chữ số của một số nguyên. Viết chương trình nhập vào một số nguyên, dùng hàm trên kiểm tra xem số đó có chia hết cho 3 không. Một số chia hết cho 3 khi tổng các chữ số của nó chia hết cho 3.

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

6. Viết chương trình tính các tổng sau:

a) $S= 1 + x + x^2 + x^3 + ... + x^n$

b) $S= 1 - x + x^2 - x^3 + ... (-1)^n x^n$

c) $S= 1 + x/1! + x^2/2! + x^3/3! + ... + x^n/n!$

Trong đó n là một số nguyên dương và x là một số bất kỳ được nhập từ bàn phím khi chạy chương trình.

7. Tam giác Pascal là một bảng số, trong đó hàng thứ 0 bằng 1, mỗi một số hạng của hàng thứ n+1 là một tổ hợp chập k của n ($C_n^k = \frac{n!}{k!(n-k)!}$)

Tam giác Pascal có dạng sau:

1 (hàng 0)
1 1 (hàng 1)
1 2 1 (hàng 2)
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1 (hàng 6)
.....

Viết chương trình hiển thị lên màn hình tam giác Pascal có n hàng (n nhập vào khi chạy chương trình) bằng cách tạo hai hàm tính giai thừa và tính tổ hợp.

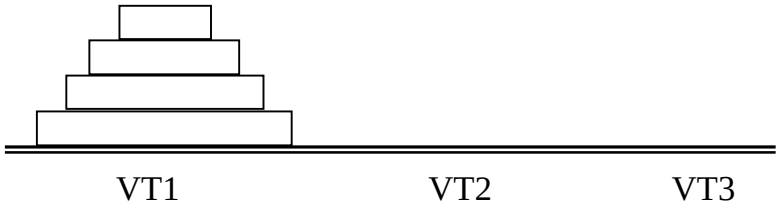
8. Yêu cầu như câu 7 nhưng dựa công thức truy hồi của tổ hợp mà viết thành 1 hàm đệ quy để tính số tổ hợp chập k của n phần tử.

$$C_n^k = \begin{cases} 1, \text{nÕu } k = 0 \text{ hoÆc } k = n \\ C_{n-1}^{k-1} + C_{n-1}^{k}, \text{nÕu } 1 < k < n \end{cases}$$

9. Viết chương trình in dãy Fibonacci đã nêu trong bằng phương pháp dùng một hàm Fibonacci F có tính đệ quy.

$$F_n = \begin{cases} 1, n\tilde{O}un = 0\ ho\text{Æ}c\ n = 1 \\ F_{n-2} + F_{n-1}, n\tilde{O}un > 1 \end{cases}$$

10. Bài toán tháp Hà Nội: Có một cái tháp gồm n tầng, tầng trên nhỏ hơn tầng dưới (hình vẽ). Hãy tìm cách chuyển cái tháp này từ vị trí thứ nhất sang vị trí thứ hai thông qua vị trí trung gian thứ ba. Biết rằng chỉ được chuyển mỗi lần một tầng và không được để tầng lớn trên tầng nhỏ.



Chương 6

MẢNG

Chương này trình bày về kiểu dữ liệu mảng - một kiểu dữ liệu có cấu trúc được sử dụng phổ biến khi lập trình. Nội dung chính của chương này gồm:

- *Khái niệm về kiểu dữ liệu mảng cũng như ứng dụng của kiểu dữ liệu này.*
- *Cách khai báo biến kiểu mảng và các phép toán trên các phần tử của mảng.*

I. GIỚI THIỆU KIỂU DỮ LIỆU MẢNG TRONG C

Mảng là một tập hợp các phần tử cố định có cùng một kiểu, gọi là kiểu phần tử. Kiểu phần tử có thể là có các kiểu bất kỳ: ký tự, số, chuỗi ký tự…; cũng có khi ta sử dụng kiểu mảng để làm kiểu phần tử cho một mảng (trong trường hợp này ta gọi là mảng của mảng hay mảng nhiều chiều).

Ta có thể chia mảng làm 2 loại: mảng 1 chiều và mảng nhiều chiều.

Mảng là kiểu dữ liệu được sử dụng rất thường xuyên. Chẳng hạn người ta cần quản lý một danh sách họ và tên của khoảng 100 sinh viên trong một lớp. Nhận thấy rằng mỗi họ và tên để lưu trữ ta cần 1 biến kiểu chuỗi, như vậy 100 họ và tên thì cần khai báo 100 biến kiểu chuỗi. Nếu khai báo như thế này thì đoạn khai báo cũng như các thao tác trên các họ tên sẽ rất dài dòng và rắc rối. Vì thế, kiểu dữ liệu mảng giúp ích ta trong trường hợp này; chỉ cần khai báo 1 biến, biến này có thể coi như là tương đương với 100 biến chuỗi ký tự; đó là 1 mảng mà các phần tử của nó là chuỗi ký tự. Hoặc 1 thí dụ khác là việc lưu trữ các từ khóa của ngôn ngữ lập trình C, ta cũng có thể dùng đến một mảng để lưu trữ chúng.

II. MẢNG 1 CHIỀU

Xét dưới góc độ toán học, mảng 1 chiều giống như một vector. Mỗi phần tử của mảng một chiều có giá trị không phải là một mảng khác.

II.1. Khai báo

II.1.1. Khai báo mảng với số phần tử xác định (khai báo tường minh)

Cú pháp: <Kiểu> <Tên mảng ><[số phần tử]>

Ý nghĩa:

- **Tên mảng**: đây là một tên đặt đúng theo quy tắc đặt tên của danh biểu. Tên này cũng mang ý nghĩa là tên biến mảng.

- **Số phần tử**: là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu (hay nói khác đi kích thước của mảng là gì).

- **Kiểu**: mỗi phần tử của mảng có dữ liệu thuộc kiểu gì.

- Ở đây, ta khai báo một biến mảng gồm có ***số phần tử*** phần tử, phần tử thứ nhất là ***tên mảng*** [***0***], phần tử cuối cùng là ***tên mảng***[***số phần tử -1***]

Thí dụ:

int a[10]; /* Khai báo biến mảng tên a, phần tử thứ nhất là a[0], phần tử cuối cùng là a[9].*/

→ Ta có thể coi mảng a là một dãy liên tiếp các phần tử trong bộ nhớ như sau:

Vị trí	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

II.1.2. Khai báo mảng với số phần tử không xác định (khai báo không tường minh)

Cú pháp: **<Kiểu> <Tên mảng> <[]>**

Khi khai báo, không cho biết rõ số phần tử của mảng, kiểu khai báo này thường được áp dụng trong các trường hợp: vừa

khai báo vừa gán giá trị, khai báo mảng là tham số hình thức của hàm.

a. Vừa khai báo vừa gán giá trị

Cú pháp:

<Kiểu> <Tên mảng> []= {Các giá trị cách nhau bởi dấu phẩy};

Nếu vừa khai báo vừa gán giá trị thì mặc nhiên C sẽ hiểu số phần tử của mảng là số giá trị mà chúng ta gán cho mảng trong cặp dấu {}. Chúng ta có thể sử dụng hàm ***sizeof()*** *để lấy số phần tử của mảng như sau:*

Số phần tử=***sizeof(tên mảng)/ sizeof(kiểu)***

b. Khai báo mảng là tham số hình thức của hàm, trong trường hợp này ta không cần chỉ định cụ thể số phần tử của mảng.

II.2 Truy xuất từng phần tử của mảng

Mỗi phần tử của mảng được truy xuất thông qua **Tên biến mảng** theo sau là **chỉ số** nằm trong cặp **dấu ngoặc vuông [ ].** Chẳng hạn a[0] là phần tử đầu tiên của mảng a được khai báo ở trên. Chỉ số của phần tử mảng là một biểu thức mà giá trị là kiểu số nguyên.

Với cách truy xuất theo kiểu này, **Tên biến mảng[Chỉ số]** *có thể coi như là một biến* có kiểu dữ liệu là **kiểu** được chỉ ra trong khai báo biến mảng.

Thí dụ 1:
int a[10];
- Phần tử đầu tiên trong mảng là a[0]
- Phần tử kế tiếp trong mảng là a[1]

…

- Phần tử cuối cùng trong mảng là a[9].

Thí dụ 2: Vừa khai báo vừa gán trị cho 1 mảng 1 chiều các số nguyên. In mảng số nguyên này lên màn hình.

```
#include <stdio.h>
#include <conio.h>
main(){
      int n,i;
      int dayso[]={66,65,69,68,67,70};
      n=sizeof(dayso)/sizeof(int); //số phần tử
      printf("\n Noi dung cua mang ");
      for (i=0;i<n;i++)
          printf("%d ",dayso[i]);
}
```

Thí dụ 3: Đổi một số nguyên dương thập phân thành số nhị phân. Việc chuyển đổi này được thực hiện bằng cách lấy số đó chia liên tiếp cho 2 cho tới khi bằng 0 và lấy các số dư theo chiều ngược lại để tạo thành số nhị phân. Trong thí dụ này, mảng một chiều được sử dụng để lưu lại các số dư đó. Chương trình cụ thể như sau:

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

main(){
     unsigned int N, Du;
     unsigned int NhiPhan[20];
     int K=0, i;
     printf("Nhap vao so nguyen N= ");
     scanf("%u",&N);
     do{
          Du=N % 2;
          NhiPhan[K]=Du;//Luu số dư vào mảng ở vị trí K
          K++;  // Tăng K lên để lần kế lưu vào vị trí kế
          N = N/2;
     } while(N>0);

     printf("Dang nhi phan la: ");
     for(i=K-1;i>=0;i--)
          printf("%u",NhiPhan[i]);
     getch();
 }
```

Thí dụ 4: Nhập vào một dãy n số và sắp xếp các số theo thứ tự tăng.

Bài toán sắp xếp là 1 bài toán có ứng dụng rộng rãi trong nhiều lĩnh vực. Để sắp xếp một dãy n số, có rất nhiều giải thuật để thực hiện. Một trong số đó được mô tả như sau:

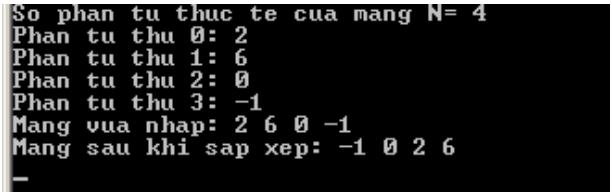
Đầu tiên đưa phần tử thứ nhất so sánh với các phần tử còn lại, nếu nó lớn hơn một phần tử đang so sánh thì đổi chỗ hai phần tử cho nhau. Sau đó tiếp tục so sánh phần tử thứ hai với các phần tử từ thứ ba trở đi ... cứ tiếp tục như vậy cho đến phần tử thứ n-1.

Chương trình sẽ được chia thành các hàm Nhap (Nhập các số), SapXep (Sắp xếp) và InMang (In các số); các tham số hình thức của các hàm này là 1 mảng không chỉ định rõ số phần tử tối đa, nhưng ta cần có *thêm số phần tử thực tế được sử dụng của mảng là bao nhiêu*, đây là một giá trị nguyên.

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

void Nhap(int a[],int N){
  int i;
  for(i=0; i< N; i++){
        printf("Phan tu thu %d: ",i);
        scanf("%d",&a[i]);
  }
}

void InMang(int a[], int N){
  int i;
  for (i=0; i<N;i++)
        printf("%d ",a[i]);
  printf("\n");
}

void SapXep(int a[], int N){
  int t, i, j;
  for(i=0;i<N-1;i++)
        for(j=i+1;j<N;j++)
              if (a[i]>a[j]){
                    t=a[i];
                    a[i]=a[j];
                    a[j]=t;
              }
}
```

```
main(){
  int b[20], N;
  printf("So phan tu thuc te N= ");
  scanf("%d",&N);
  Nhap(b,N);
  printf("Mang vua nhap: ");
  InMang(b,N);
  SapXep(b,N); // Gọi hàm sắp xếp
  printf("Mang sau khi sap xep: ");
  InMang(b,N);
  getch();
}
```

Kết quả chạy chương trình có thể là:



III. MẢNG NHIỀU CHIỀU

Mảng nhiều chiều là 1 mảng mà mỗi phần tử của mảng là 1 mảng khác.

Người ta thường sử dụng mảng nhiều chiều để lưu các ma trận, các tọa độ 2 chiều, 3 chiều…

Phần dưới đây là các vấn đề liên quan đến mảng 2 chiều; các mảng 3, 4,… chiều thì tương tự (chỉ cần tổng quát hóa lên).

III.1 Khai báo

III.1.1. Khai báo mảng 2 chiều tường minh

Cú pháp:

<Kiểu> <Tên mảng><[Số phần tử chiều 1]><[Số phần tử chiều 2]>

Thí dụ: Người ta cần lưu trữ thông tin của một ma trận gồm các số thực. Lúc này ta có thể khai báo một mảng 2 chiều như sau:

```
float m[5][6];   // Khai báo mảng 2 chiều có 5*6 phần tử là số thực
```

Trong trường hợp này, ta đã khai báo cho một ma trận có tối đa là 5 dòng, mỗi dòng có tối đa là 6 cột. Hình ảnh của ma trận này được thể hiện trong hình bên dưới:

Dòng\Cột	**0**	**1**	**2**	**3**	**4**	**5**
0	m[0][0]	m[0][1]	m[0][2]	m[0][3]	m[0][4]	m[0][5]
1	m[1][0]	m[1][1]	m[1][2]	m[1][3]	m[1][4]	m[1][5]
2	m[2][0]	m[2][1]	m[2][2]	m[2][3]	m[2][4]	m[2][5]
3	m[3][0]	m[3][1]	m[3][2]	m[3][3]	m[3][4]	m[3][5]
4	m[4][0]	m[4][1]	m[4][2]	m[4][3]	m[4][4]	m[4][5]

III.1.2. Khai báo mảng 2 chiều không tường minh

Để khai báo mảng 2 chiều không tường minh, ta vẫn phải chỉ ra số phần tử của chiều thứ hai (chiều cuối cùng).

Cú pháp:

<Kiểu> <Tên mảng> <[]><[Số phần tử chiều 2]>

Cách khai báo này cũng được áp dụng trong trường hợp vừa khai báo, vừa gán trị hay đặt mảng 2 chiều là tham số hình thức của hàm.

III.2 Truy xuất từng phần tử của mảng 2 chiều

Ta có thể truy xuất một phần tử của mảng hai chiều bằng cách viết ra **tên mảng** theo sau là hai chỉ số đặt trong hai cặp dấu ngoặc vuông. Chẳng hạn ta viết m[2][3].

Với cách truy xuất theo cách này, **Tên mảng[Chỉ số 1][Chỉ số 2]** có thể coi là 1 biến có kiểu được chỉ ra trong khai báo biến mảng.

Thí dụ 1: Viết chương trình cho phép nhập 2 ma trận a, b có m dòng n cột, thực hiện phép toán cộng hai ma trận a,b và in ma trận kết quả lên màn hình.

Trong thí dụ này, ta sẽ sử dụng hàm để chương trình ngắn gọn hơn. Trong trường hợp này, các hàm sau được định nghĩa: nhập 1 ma trận từ bàn phím, hiển thị ma trận lên màn hình, cộng 2 ma trận.

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

void Nhap(float a[][10],int M,int N){
      int i, j;
      for(i=0;i<M;i++)
            for(j=0; j<N; j++){
                printf("Phan tu o dong %d cot %d: ",i,j);
                scanf("%f",&a[i][j]);
            }
}

void InMaTran(float a[][10], int M, int N){
      int i, j;
      for(i=0;i<M;i++)
      {
            for(j=0; j< N; j++)
                  printf("%.3f    ",a[i][j]);
            printf("\n");
      }
}

// Cong 2 ma tran A & B ket qua la ma tran C
void CongMaTran(float a[][10],float b[][10],
                int M, int N,float c[][10])
{
      int i, j;
      for(i=0;i<M;i++)
            for(j=0; j<N; j++)
                  c[i][j]=a[i][j]+b[i][j];
}

main()
{
      float a[10][10], b[10][10], c[10][10];
      int M, N;
      printf("So dong M= "); scanf("%d",&M);
```

```
      printf("So cot M= "); scanf("%d",&N);
      printf("Nhap ma tran A\n");
      Nhap(a,M,N);
      printf("Nhap ma tran B\n");
      Nhap(b,M,N);
      printf("Ma tran A: \n");
      InMaTran(a,M,N);
      printf("Ma tran B: \n");
      InMaTran(b,M,N);

      CongMaTran(a,b,M,N,c);
      printf("Ma tran tong C:\n");
      InMaTran(c,M,N);
      getch();
}
```

Thí dụ 2: Nhập vào một ma trận 2 chiều gồm các số thực, in ra tổng của các phần tử trên đường chéo chính của ma trận này.

Ta nhận thấy rằng nếu ma trận a có M dòng, N cột thì các phần tử của đường chéo chính là các phần tử có dạng: a[i][i] với i $\in$ [1…min(M,N)].

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

      main(){
            float a[10][10];
            int M, N, i, j, Min;
            float T ;
            printf("So dong? ");scanf("%d",&M);
            printf("So cot? ");scanf("%d",&N);

            for(i=0;i<M;i++)
                  for(j=0; j<N; j++){
                    printf("Phan tu (%d, %d):  ",i,j);
                    scanf("%f",&a[i][j]);
                  }

            printf("Ma tran vua nhap: \n");
            for(i=0;i<M;i++){
                  for(j=0; j< N; j++)
                        printf("%.2f    ",a[i][j]);
                  printf("\n");
```

```
            }

            T=0.0;
            Min=(M>N) ? N: M; /*Tìm giá trị nhỏ nhất
của M & N */
            for(i=0;i<Min;i++)
                  T=T+a[i][i];

            printf("Tong cac phan tu o duong cheo
                        chinh la: %.2f",T);
            getch();
      }
```

IV. BÀI TẬP

1. Viết chương trình nhập vào một dãy n số thực a[0], a[1],..., a[n-1], sắp xếp dãy số theo thứ tự giảm dần. In dãy số sau khi sắp xếp.

2. Giả sử mảng a ban đầu đã được sắp thứ tự tăng dần. Viết chương trình cho phép loại bỏ các phần tử trùng nhau trong mảng a.

Thí dụ: Mảng a -1 0 0 2 2 3

Mảng sau khi loại các phần tử trùng nhau:

-1 0 2 3

3. Viết chương trình nhập vào một mảng, hãy xuất ra màn hình:

- Phần tử lớn nhất của mảng.
- Phần tử nhỏ nhất của mảng.
- Tính tổng của các phần tử trong mảng.
- Tính tổng bình phương các số âm trong mảng.

4. Viết chương trình nhập vào một mảng số tự nhiên. Hãy xuất ra màn hình:

- Dòng 1: gồm các số lẻ, tổng cộng có bao nhiêu số lẻ.
- Dòng 2: gồm các số chẵn, tổng cộng có bao nhiêu số chẵn.
- Dòng 3 : gồm các số nguyên tố, tổng cộng có bao nhiêu số nguyên tố.

- Dòng 4 : gồm các số không phải là số nguyên tố, tổng cộng có bao nhiêu số không nguyên tố.

5. Viết chương trình thực hiện việc đảo một mảng một chiều.

Thí dụ : 1 2 3 4 5 7 9 đảo thành 9 7 5 4 3 2 1 .

6. Viết chương trình nhập vào một dãy các số theo thứ tự tăng, nếu nhập sai quy cách thì yêu cầu nhập lại. In dãy số sau khi đã nhập xong. Nhập thêm một số mới và chèn số đó vào dãy đã có sao cho dãy vẫn đảm bảo thứ tự tăng. In lại dãy số để kiểm tra.

7. Viết chương trình thực hiện việc trộn hai mảng có thứ tự thành một mảng có thứ tự. Yêu cầu không được gộp 2 mảng lại rồi mới sắp thứ tự nhờ vào 1 hàm sắp xếp.

8. Viết chương trình nhập vào một ma trận (mảng hai chiều) các số nguyên, gồm m hàng, n cột. In ma trận đó lên màn hình. Nhập một số nguyên khác vào và xét xem có phần tử nào của ma trận trùng với số này không ? Ở vị trí nào ? Có bao nhiêu phần tử ?

9. Viết chương trình để chuyển đổi vị trí từ dòng thành cột của một ma trận (ma trận chuyển vị) vuông 4 hàng 4 cột.

Thí dụ:

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

Viết chương trình cho ma trận trong trưởng hợp tổng quát (cấp m*n: m dòng, n cột).

10. Viết chương trình nhập vào hai ma trận A có cấp m, k và B có cấp k, n. In hai ma trận lên màn hình. Tích hai ma trận A và B là ma trận C được tính bởi công thức:

$$c_{ij} = a_{i1} * b_{1j} + a_{i2} * b_{2j} + a_{i3} * b_{3j} + ... + a_{ik} * b_{kj}$$
$$(i=0,1,2,...m-1;j=0,1,2...n-1)$$

Tính ma trận tích C và in kết quả lên màn hình.

11. Xét ma trận A vuông cấp n, các phần tử a[i, i] (i= 1 ... n) được gọi là đường chéo chính của ma trận vuông A. Ma trận

vuông A được gọi là ma trận tam giác nếu tất cả các phần tử dưới đường chéo chính đều bằng 0. Định thức của ma trận tam giác bằng tích các phần tử trên đường chéo chính.

Ta có thể chuyển một ma trận vuông bất kỳ về ma trận tam giác bằng thuật toán:

- Xét cột i (i =0,1...n-2)
- Trong cột i xét các phần tử a[k,i] (k=i+1…n-1)
    + Nếu a[k,i]=0 thì tăng k lên xét phần tử khác
    + Nếu a[k,i] <> 0 thì làm như sau:
        Nhân toàn bộ hàng k với - a[i,i]/a[k,i]
        Lấy hàng k cộng vào hàng i sau khi thực hiện phép nhân trên.
        Đổi chỗ hai hàng i và k cho nhau
        Nhân toàn bộ hàng k với -1 sau khi đã đổi chỗ với hàng i
        Tăng k lên xét phần tử khác.

Viết chương trình tính định thức cấp n thông qua các bước nhập ma trận, in ma trận, đưa ma trận về dạng tam giác, in ma trận tam giác, in kết quả tính định thức.

Chương 7

CON TRỎ

Các vấn đề được trình bày trong chương này:

- *Khái niệm về kiểu dữ liệu "con trỏ".*
- *Cách khai báo và cách sử dụng biến kiểu con trỏ.*
- *Mối quan hệ giữa mảng và con trỏ.*

I. GIỚI THIỆU KIỂU DỮ LIỆU CON TRỎ

Các biến chúng ta đã biết và sử dụng trước đây đều là biến có kích thước và kiểu dữ liệu xác định. Người ta gọi các biến kiểu này là biến tĩnh. Khi khai báo biến tĩnh, một lượng ô nhớ cho các biến này sẽ được cấp phát mà không cần biết trong quá trình thực thi chương trình có sử dụng hết lượng ô nhớ này hay không. Mặt khác, các biến tĩnh dạng này sẽ tồn tại trong suốt thời gian thực thi chương trình dù có những biến mà chương trình chỉ sử dụng 1 lần rồi bỏ.

Do đó, một số vấn đề có thể gặp phải khi sử dụng các biến tĩnh:

- Cấp phát ô nhớ dư, gây ra lãng phí ô nhớ.
- Cấp phát ô nhớ thiếu, chương trình thực thi bị lỗi.

Để tránh những hạn chế trên, người lập trình được cung cấp một loại biến đặc biệt gọi là biến động với các đặc điểm sau:

- Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình.
- Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được cấp phát cho biến có thể thay đổi.
- Sau khi sử dụng xong có thể giải phóng để tiết kiệm chỗ trong bộ nhớ.

Tuy nhiên một số ngôn ngữ lập trình chẳng hạn như C không cung cấp 1 cách thức trực tiếp để thao tác với biến động. Vì thế,

ngôn ngữ C cung cấp cho ta một loại biến đặc biệt là biến con trỏ (pointer) với các đặc điểm:

- Biến con trỏ không chứa dữ liệu mà chỉ chứa địa chỉ của dữ liệu hay chứa địa chỉ của ô nhớ chứa dữ liệu. Lúc đó ô nhớ chứa dữ liệu có thể coi như là 1 biến động.
- Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu, luôn có kích thước cố định là 2 byte (giá trị nguyên để chứa địa chỉ).

Như vậy thực chất vai trò của biến con trỏ chính là hỗ trợ cho người lập trình cách thức thao tác với các biến động.

II. KHAI BÁO VÀ SỬ DỤNG BIẾN CON TRỎ

II.1. Khai báo biến con trỏ

Cú pháp: <Kiểu> * <Tên con trỏ>

Ý nghĩa: Khai báo một biến có tên là ***Tên con trỏ*** dùng để chứa địa chỉ của các biến có kiểu ***Kiểu.***

Thí dụ 1: Khai báo 2 biến a,b có kiểu int và 2 biến pa, pb là 2 biến con trỏ kiểu int.

```
int a, b, *pa, *pb;
```

Thí dụ 2: Khai báo biến f kiểu float và biến pf là con trỏ float

```
float  f, *pf;
```

Ghi chú: Nếu chưa muốn khai báo kiểu dữ liệu mà con trỏ ptr đang chỉ đến, ta sử dụng con trỏ void:

```
void *ptr;
```

Sau đó, nếu ta muốn con trỏ ptr chỉ đến kiểu dữ liệu gì cũng được. Tác dụng của khai báo này là chỉ dành ra 2 bytes trong bộ nhớ để cấp phát cho biến con trỏ ptr.

II.2. Các thao tác trên con trỏ

II.2.1 Gán địa chỉ của biến cho biến con trỏ

Toán tử & dùng để trả về địa chỉ của 1 biến (định vị con trỏ đến địa chỉ của một biến).

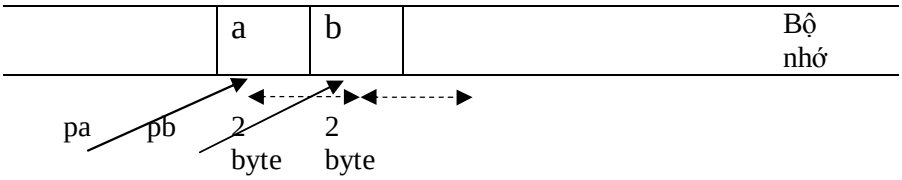
Cú pháp: <Tên biến con trỏ>=&<Tên biến>

Giải thích: Ta gán địa chỉ của biến ***Tên biến*** cho con trỏ ***Tên biến con trỏ***.

Thí dụ: Gán địa chỉ của biến a cho con trỏ pa, gán địa chỉ của biến b cho con trỏ pb.

pa=&a; pb=&b;

Lúc này, hình ảnh của các biến trong bộ nhớ được mô tả:



Lưu ý:

Khi gán địa chỉ của 1 biến cho 1 con trỏ cần phải lưu ý kiểu dữ liệu của chúng. Ví dụ sau đây không đúng do không tương thích kiểu:

```
int  Bien_Nguyen;
float  *Con_Tro_Thuc;
...
Con_Tro_Thuc=&Bien_Nguyen;
```

Phép gán ở đây là sai vì Con_Tro_Thuc là một con trỏ kiểu float (nó chỉ có thể chứa được địa chỉ của biến kiểu float); trong khi đó, Bien_Nguyen có kiểu int, nghĩa là &Bien_Nguyen chính là địa chỉ của 1 biến int..

II.2.2 Nội dung của ô nhớ con trỏ chỉ tới

Để truy cập đến nội dung của ô nhớ mà con trỏ chỉ tới, ta sử dụng cú pháp:

***<Tên biến con trỏ>**

Với cách truy cập này thì ***<Tên biến con trỏ>** có thể coi là một biến có kiểu được mô tả trong phần khai báo biến con trỏ. Để ý rằng cách viết như trên chính là cách thức thao tác với các biến động.

Thí dụ: Khai báo, gán địa chỉ cũng như lấy nội dung vùng nhớ của biến con trỏ:

```
int x=100;
int *ptr;
ptr=&x;
int y= *ptr;
```

Lưu ý: Khi gán địa chỉ của một biến cho một biến con trỏ, mọi sự thay đổi trên nội dung ô nhớ con trỏ chỉ tới sẽ làm giá trị của biến thay đổi theo (thực chất nội dung ô nhớ và biến chỉ là một).

Thí dụ: Đoạn chương trình sau thấy rõ sự thay đổi này :

```
#include <stdio.h>
main(){
      int a,b,*pa,*pb;
      a=2;
      b=3;
      printf("\nGia tri cua bien a=%d \n
                  Gia tri cua bien b=%d ",a,b);
      pa=&a;
      pb=&b;
      printf("\nNoi dung cua o nho con tro pa
                  tro toi=%d",*pa);
      printf("\nNoi dung cua o nho con tro pb
                  tro toi=%d ",*pb);
      *pa=20; // Thay đổi giá trị của *pa
      *pb=20; // Thay đổi giá trị của *pb
      printf("\nGia tri moi cua bien a=%d \n
            Gia tri moi cua bien b=%d ",a,b);
}
```

Kết quả thực hiện chương trình:

```
Gia tri cua bien a=2
Gia tri cua bien b=3
Noi dung cua o nho con tro pa tro toi=2
Noi dung cua o nho con tro pb tro toi=3
Gia tri moi cua bien a=20
Gia tri moi cua bien b=20
```

II.2.3 Cấp phát vùng nhớ để biến con trỏ quản lý địa chỉ

Thực chất của con trỏ là chứa địa chỉ trong bộ nhớ của 1 biến khác (thông thường gọi là 1 biến động). Đặc trưng của biến động chính là chúng được cấp phát 1 cách động trong quá trình thực hiện của chương trình. Ngôn ngữ C hỗ trợ một số hàm để cấp phát vùng nhớ cho các biến động này; kết quả trả về của các hàm cấp phát này là địa chỉ bắt đầu (con trỏ) của vùng nhớ được cấp phát. Đó là các hàm malloc(), calloc() trong thư viện malloc.h.

Cú pháp các hàm:

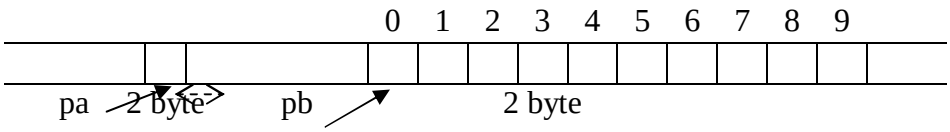
- void *malloc(size_t size): Cấp phát vùng nhớ có kích thước là size.

- void *calloc(size_t nitems, size_t size): Cấp phát vùng nhớ có kích thước là nitems*size.

Thí dụ: Giả sử ta có khai báo:

```
int a, *pa, *pb;
pa = (int*)malloc(sizeof(int)); /* Cấp phát vùng nhớ có kích thước bằng với kích thước của một số nguyên */
pb= (int*)calloc(10, sizeof(int)); /* Cấp phát vùng nhớ có thể chứa được 10 số nguyên*/
```

Lúc này hình ảnh trong bộ nhớ như sau:



Lưu ý: Khi sử dụng hàm malloc() hay calloc(), ta phải ép kiểu vì nguyên mẫu (prototype) của các hàm này trả về con trỏ kiểu void.

II.3.4 Cấp phát lại vùng nhớ cho biến con trỏ

Trong quá trình thao tác trên biến con trỏ, nếu ta cần cấp phát thêm vùng nhớ có kích thước lớn hơn vùng nhớ đã cấp phát, ta sử dụng hàm realloc().

Cú pháp: void *realloc(void *block, size_t size)

Ý nghĩa:

- Cấp phát lại 1 vùng nhớ cho con trỏ *block* quản lý, vùng nhớ này có kích thước mới là *size*; khi cấp phát lại thì nội dung vùng nhớ trước đó vẫn tồn tại.

- Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới. Địa chỉ này có thể khác với địa chỉ được chỉ ra khi cấp phát ban đầu.

Thí dụ: Trong thí dụ trên ta có thể cấp phát lại vùng nhớ do con trỏ pa quản lý như sau:

```
int a, *pa;
pa = (int*)malloc(sizeof(int)); /* Cấp phát vùng nhớ có kích thước 2 byte */
pa = realloc(pa, 6);  /* Cấp phát lại vùng nhớ có kích thước 6 byte */
```

II.3.5 Giải phóng vùng nhớ cho biến con trỏ

Một vùng nhớ đã cấp phát được quản lý bởi 1 biến con trỏ, khi không còn sử dụng nữa, ta sẽ thu hồi lại vùng nhớ này nhờ hàm free().

Cú pháp: void free(void *block)

Ý nghĩa: Giải phóng vùng nhớ được quản lý bởi con trỏ block.

Thí dụ: Ở thí dụ trên, sau khi thực hiện xong, ta giải phóng vùng nhớ cho 2 biến con trỏ pa & pb:

```
free(pa); free(pb);
```

II.3.6 Một số phép toán trên con trỏ

a. Phép gán con trỏ: Hai con trỏ cùng kiểu có thể gán cho nhau.

Thí dụ:
```
int a, *p, *a ; float *f;
a = 5 ; p = &a ; q = p ; /* đúng */
f = p ; /* sai do khác kiểu */
```

Ta cũng có thể ép kiểu con trỏ theo cú pháp:

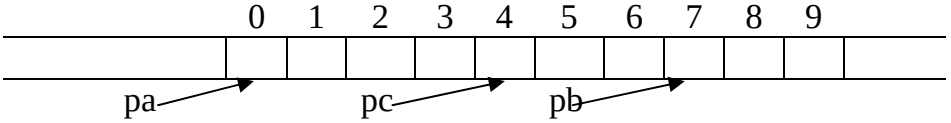
(<Kiểu kết quả>*)<Tên con trỏ>

Chẳng hạn, thí dụ trên được viết lại:
```
int a, *p, *a ; float *f;
a = 5 ; p = &a ; q = p ; /* đúng */
f = (float*)p; // Đúng nhờ ép kiểu
```

b. Cộng, trừ con trỏ với một số nguyên: Ta có thể cộng (+), trừ (-) 1 con trỏ với 1 số nguyên N nào đó; kết quả trả về là 1 con trỏ. Con trỏ này chỉ đến vùng nhớ cách vùng nhớ của con trỏ hiện tại N phần tử.

Thí dụ: Cho đoạn chương trình sau:
```
int *pa;
pa = (int*) malloc(20); /* Cấp phát vùng nhớ 20
byte=10 số nguyên */
int *pb, *pc;
pb = pa + 7;
pc = pb - 3;
```

Lúc này hình ảnh của pa, pb, pc như sau:



c. Con trỏ NULL: là con trỏ không chứa địa chỉ nào cả. Ta có thể gán giá trị NULL cho 1 con trỏ có kiểu bất kỳ.

d. Lưu ý:

- Ta không thể cộng 2 con trỏ với nhau.

- Phép trừ 2 con trỏ cùng kiểu sẽ trả về 1 giá trị nguyên (int). Đây chính là khoảng cách (số phần tử) giữa 2 con trỏ đó. *Chẳng hạn*, trong ví dụ trên pc-pa=4.

III. CON TRỎ VÀ MẢNG

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Trong C, thực chất tên mảng chính là con trỏ chứa địa chỉ của phần tử đầu tiên trong mảng đó. Do đó, những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác lập qua biến con trỏ.

III.1 Con trỏ và mảng 1 chiều

III.1.1 Truy cập các phần tử mảng theo dạng con trỏ

Ta có các quy tắc sau:

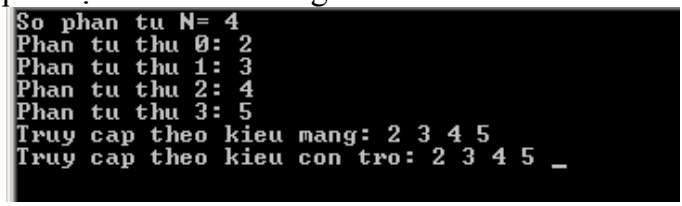
&<Tên mảng>[0] *tương đương với* ***<Tên mảng>***
&<Tên mảng> [<Vị trí>] *tương đương với* ***<Tên mảng> + <Vị trí>***
<Tên mảng>[<Vị trí>] *tương đương với* ****(<Tên mảng> + <Vị trí>)***

Thí dụ: Cho 1 mảng 1 chiều các số nguyên a có 5 phần tử, truy cập các phần tử theo kiểu mảng và theo kiểu con trỏ.

```
#include <stdio.h>
#include <conio.h>
   // Nhập mảng bình thường
void NhapMang(int a[], int N){
   int i;
   for(i=0;i<N;i++)
   {
         printf("Phan tu thu %d: ",i);
         scanf("%d",&a[i]);
   }
}
   // Nhập mảng theo dạng con trỏ
void NhapContro(int a[], int N){
   int i;
   for(i=0;i<N;i++)
   {
         printf("Phan tu thu %d: ",i);
         scanf("%d",a+i);
   }
}

main()
{
   int a[20],N, i;
   printf("So phan tu N= ");scanf("%d",&N);
   NhapMang(a,N); // NhapContro(a,N)
```

```
      printf("Truy cap theo kieu mang: ");
      for(i=0;i<N;i++)
            printf("%d ",a[i]);
      printf("\nTruy cap theo kieu con tro: ");
      for(i=0;i<N;i++)
            printf("%d ",*(a+i));
      getch();
      }
```

Kết quả thực thi của chương trình:



III.1.2 Truy xuất từng phần tử đang được quản lý bởi con trỏ theo dạng mảng

<Tên biến>[<Vị trí>] *tương đương với* ****(<Tên biến> + <Vị trí>)***

&<Tên biến>[<Vị trí>] *tương đương với* ***(<Tên biến> + <Vị trí>)***

Trong đó <Tên biến> là biến con trỏ, <Vị trí> là 1 biểu thức số nguyên.

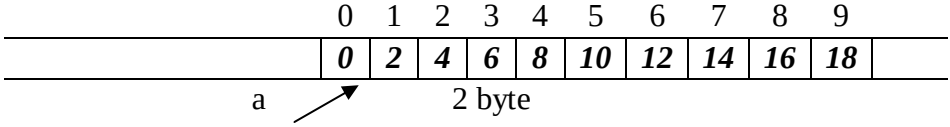
Thí dụ: Giả sử có khai báo:

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

main(){
   int *a;
   int i;
   a=(int*)malloc(sizeof(int)*10);
   for(i=0;i<10;i++)
         a[i] = 2*i;
   printf("Truy cap theo kieu mang: ");
   for(i=0;i<10;i++)
         printf("%d ",a[i]);
   printf("\nTruy cap theo kieu con tro: ");
   for(i=0;i<10;i++)
         printf("%d ",*(a+i));
   getch();
}
```

Kết quả chương trình:

```
Truy cap theo kieu mang: 0 2 4 6 8 10 12 14 16 18
Truy cap theo kieu con tro: 0 2 4 6 8 10 12 14 16 18
```

Với khai báo ở trên, hình ảnh của con trỏ a trong bộ nhớ:



III.1.3 Con trỏ chỉ đến phần tử mảng

Giả sử con trỏ ptr chỉ đến phần tử a[i] nào đó của mảng a thì:

ptr + j *chỉ đến phần tử thứ j sau a[i], tức* ***a[i+j]***

ptr - j *chỉ đến phần tử đứng trước a[i], tức* ***a[i-j]***

Thí dụ: Giả sử có 1 mảng mang_int, cho con trỏ contro_int chỉ đến phần tử thứ 5 trong mảng. In ra các phần tử của contro_int & mang_int.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
main(){
 int i,mang_int[10];
 int *contro_int;
 for(i=0;i<=9;i++)
       mang_int[i]=i*2;

 contro_int=&mang_int[5];
 printf("\nNoi dung cua mang_int ban dau=");
 for (i=0;i<=9;i++)
       printf("%d  ",mang_int[i]);

 printf("\nNoi dung cua contro_int ban dau =");
 for (i=0;i<5;i++)
       printf("%d  ",contro_int[i]);

 for(i=0;i<5;i++)
       contro_int[i]++;
 printf("\n-----------------------------------");
```

```
    printf("\nNoi dung cua mang_int sau khi tang 1=");
    for (i=0;i<=9;i++)
          printf("%d  ",mang_int[i]);

    printf("\nNoi dung cua contro_int
                        sau khi tang 1=");
    for (i=0;i<5;i++)
          printf("%d  ",contro_int[i]);

    if (contro_int!=NULL)
     free(contro_int);
    getch();
}
```

Kết quả chương trình

```
Noi dung cua mang_int ban dau=0  2  4  6  8  10  12  14  16  18
Noi dung cua contro_int ban dau =10  12  14  16  18
--------------------------------------------------
Noi dung cua mang_int sau khi tang 1=0  2  4  6  8  11  13  15  17  19
Noi dung cua contro_int sau khi tang 1=11  13  15  17  19
```

III.2 Con trỏ và mảng nhiều chiều

Giả sử ta có mảng 2 chiều và biến con trỏ như sau:

int a[n][m];

int *contro_int;

Thực hiện phép gán contro_int=a;

Khi đó phần tử a[0][0] được quản lý bởi contro_int;
a[0][1] được quản lý bởi contro_int+1;
a[0][2] được quản lý bởi contro_int+2;
...
a[1][0] được quản lý bởicontro_int+m;
a[1][1] được quản lý bởi contro_int+m+1;
...
a[n][m] được quản lý bởi contro_int+n*m;

Tương tự như thế đối với mảng nhiều hơn 2 chiều.

Thí dụ: Sự tương đương giữa mảng 2 chiều và con trỏ.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
main()
{
  int i,j;
  int mang_int[4][5]={1,2,3,4,5,6,7,8,9,10,11,
                      12,13,14,15,16,17,18,19,20};
  int *contro_int;
  contro_int=(int*)mang_int;

  printf("\nNoi dung cua mang_int ban dau=");




  for (i=0;i<4;i++)
  {
        printf("\n");
        for (j=0;j<5;j++)
              printf("%d\t",mang_int[i][j]);
  }

  printf("\n---------------------------------");
  printf("\nNoi dung cua contro_int ban dau \n");
  for (i=0;i<20;i++)
        printf("%d ",contro_int[i]);

  for(i=0;i<20;i++)
        contro_int[i]++;

  printf("\n-------------------------------------");
  printf("\nNoi dung cua mang_int sau khi tang 1=");
  for (i=0;i<4;i++)
  {
        printf("\n");
        for (j=0;j<5;j++)
              printf("%d\t",mang_int[i][j]);
  }

  printf("\nNoi dung cua contro_int
                    sau khi tang 1=\n");

  for (i=0;i<20;i++)
        printf("%d ",contro_int[i]);
  if (contro_int!=NULL)
   free(contro_int);
  getch();
```

```
}
```

Kết quả thực hiện chương trình như sau:

```
Noi dung cua mang_int ban dau=
1       2       3       4       5
6       7       8       9       10
11      12      13      14      15
16      17      18      19      20
--------------------------------
Noi dung cua contro_int ban dau
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
------------------------------------------------------------
Noi dung cua mang_int sau khi tang 1=
2       3       4       5       6
7       8       9       10      11
12      13      14      15      16
17      18      19      20      21
Noi dung cua contro_int sau khi tang 1=
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
```

IV. CON TRỎ LÀ THAM SỐ HÌNH THỨC CỦA HÀM

Khi tham số hình thức của hàm là một con trỏ thì theo nguyên tắc gọi hàm ta dùng tham số thực tế là 1 con trỏ có kiểu giống với kiểu của tham số hình thức. Nếu lúc thực thi hàm ta có sự thay đổi trên nội dung vùng nhớ được chỉ bởi con trỏ tham số hình thức thì lúc đó nội dung vùng nhớ được chỉ bởi tham số thực tế cũng sẽ bị thay đổi theo.

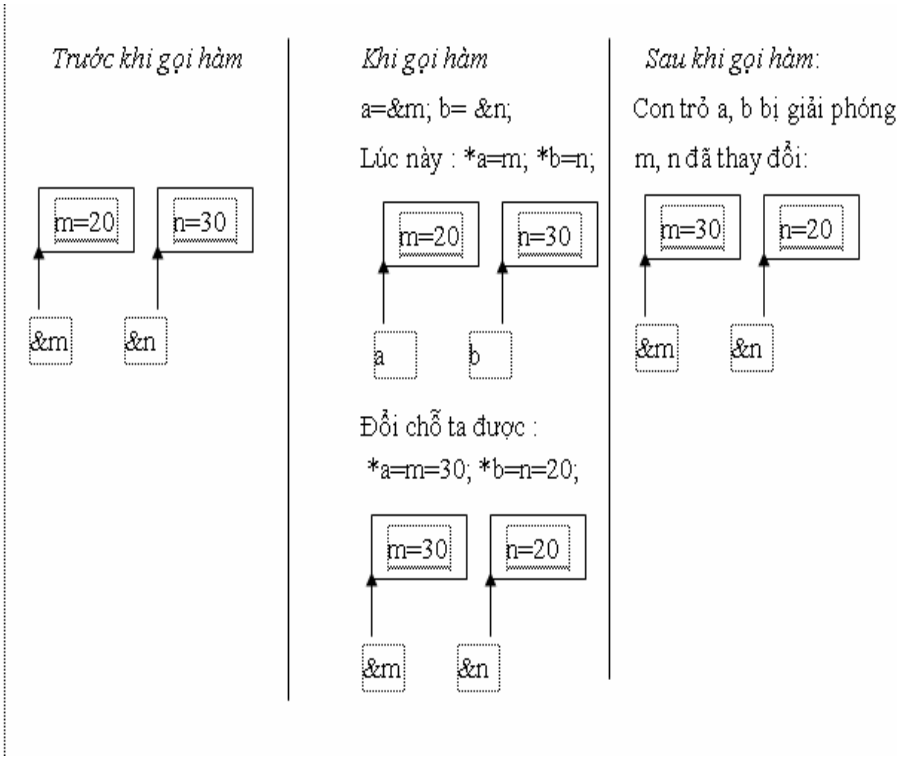
Thí dụ : Xét hàm hoán vị được viết như sau :

```
#include<stdio.h>
#include<conio.h>
void HoanVi(int *a, int *b){
     int c=*a;
     *a=*b;
     *b=c;
}

main(){
     int m=20,n=30;
     printf("Truoc khi goi ham m= %d,
                n= %d\n",m,n);
     HoanVi(&m,&n);
     printf("Sau khi goi ham m= %d,
                n= %d",m,n);
     getch();
}
```

Kết quả thực thi chương trình:

```
Truoc khi goi ham m= 20, n= 30
Sau khi goi ham m= 30, n= 20
```

Giải thích:



Với việc khi tham số hình thức là 1 con trỏ có thể làm thay đổi giá trị của vùng dữ liệu của con trỏ tham số thực tế nên người ta có thể sử dụng cách thức truyền tham số là con trỏ để có thể nhận kết quả trả về là vùng dữ liệu của con trỏ tham số thực tế.

Thí dụ: Viết 1 hàm tính n! nhưng sử dụng tham số hình thức là 1 con trỏ để có thể nhận được kết quả trả về sau khi thực thi hàm :

```
void giaithua(int n, long *K){
      int i;
      (*K)=1L;
      for (i=1;i<=n;i++)
            (*K)=(*K)*i;
}

main(){
      int n;
      long KQ;
      printf("Nhap so n = ");scanf("%d",&n);
      giaithua(n,&KQ);
      printf("\n%d!= %ld",n,KQ);
      getch();
}
```

Chương 8

CHUỖI KÝ TỰ

Các vấn đề được trình bày trong chương này :

- *Khái niệm về chuỗi ký tự.*
- *Một số hàm xử lý chuỗi và áp dụng của chúng.*

I. KHÁI NIỆM

Chuỗi ký tự là một dãy gồm các ký tự hoặc một mảng các ký tự được kết thúc bằng ký tự '\0' (còn được gọi là ký tự NULL trong bảng mã ASCII).

Các hằng chuỗi ký tự được đặt trong cặp dấu nháy kép "".

II. KHAI BÁO

Ngôn ngữ C không hỗ trợ chuỗi ký tự như là 1 kiểu riêng. Thực chất chuỗi ký tự trong C được coi như là 1 mảng gồm các phần tử là 1 ký tự (kiểu char) với ký tự cuối cùng là ký tự có mã ASCII là 0 ('\0'). Vì thế, việc khai báo chuỗi ký tự trong C chính là khai báo 1 mảng ký tự (hoặc có thể là 1 con trỏ chỉ đến vùng nhớ 1 ký tự).

II.1 Khai báo theo mảng

Cú pháp: char <Biến> [Chiều dài tối đa]

Thí dụ: Trong chương trình, ta có khai báo:

```
char Ten[12];
```

Đây là khai báo của 1 biến chuỗi Ten có chiều dài tối đa 12 ký tự.

Ghi chú:

- Chiều dài tối đa không nên khai báo thừa để tránh lãng phí bộ nhớ, nhưng cũng không nên khai báo thiếu.

- Với việc khai báo 1 mảng ký tự như trên, ký tự '\0' không được tự động thêm vào cuối chuỗi.

II.2 Khai báo theo con trỏ

Cú pháp: char *<Biến>

Thí dụ: Trong chương trình, ta có khai báo:

char *Ten;

Trong khai báo này, bộ nhớ sẽ dành 2 byte để lưu trữ địa chỉ của biến con trỏ Ten đang chỉ đến, chưa cung cấp nơi để lưu trữ dữ liệu. Muốn có chỗ để lưu trữ dữ liệu, ta phải gọi đến hàm *malloc()* hoặc *calloc()* có trong "malloc.h", sau đó mới gán dữ liệu cho biến.

II.3 Vừa khai báo vừa gán giá trị

Cú pháp: char <Biến>[]=<"Hằng chuỗi">

Thí dụ:

```
#include<stdio.h>
#include<conio.h>
main(){
   char Chuoi[]="Mau nang hay la mau mat em" ;
   printf("Vua khai bao vua gan trị : %s",
           Chuoi) ;
   getch();
}
```

Ghi chú: Chuỗi được khai báo là một mảng các ký tự nên các thao tác trên mảng có thể áp dụng đối với chuỗi ký tự.

III. CÁC THAO TÁC TRÊN CHUỖI KÝ TỰ

III.1. Nhập xuất chuỗi

III.1.1 Nhập chuỗi từ bàn phím

Để nhập một chuỗi ký tự từ bàn phím, ta sử dụng hàm gets()

Cú pháp: **gets(<Biến chuỗi>)**

Thí dụ: char Ten[20];

gets(Ten);

Ta cũng có thể sử dụng hàm scanf() để nhập dữ liệu cho biến chuỗi, tuy nhiên lúc này ta chỉ có thể nhập được một chuỗi không có dấu khoảng trắng.

Khi nhập chuỗi bằng hàm gets, ký tự '\0' được tự động thêm vào cuối chuỗi.

III.1.2 Xuất chuỗi lên màn hình

Để xuất một chuỗi (biểu thức chuỗi) lên màn hình, ta sử dụng hàm puts().

Cú pháp: **puts(<Biểu thức chuỗi>)**

Thí dụ: Nhập vào một chuỗi và hiển thị trên màn hình chuỗi vừa nhập.

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

main(){
      char Ten[12];
      printf("Nhap chuoi: ");gets(Ten);
      printf("Chuoi vua nhap: ");puts(Ten);
      getch();
}
```

Ngoài ra, ta có thể sử dụng hàm printf(), cputs() (trong conio.h) để hiển thị chuỗi lên màn hình.

III.2 Một số hàm xử lý chuỗi (trong string.h)

III.2.1 Cộng chuỗi - Hàm strcat()

Cú pháp: char *strcat(char *des, const char *source)

Hàm này có tác dụng ghép chuỗi nguồn vào chuỗi đích.

Thí dụ: Nhập vào họ lót và tên của một người, sau đó in cả họ và tên của họ lên màn hình.

```
#include<stdio.h>
#include<string.h>
main(){
 char HoLot[30], Ten[12];
 printf("Nhap Ho Lot: ");gets(HoLot);
 printf("Nhap Ten: ");gets(Ten);
 strcat(HoLot,Ten); // Ghep Ten vao HoLot
 printf("Ho ten la: ");puts(HoLot);
}
```

III.2.2 Xác định độ dài chuỗi - Hàm strlen()

Cú pháp: int strlen(const char* s)

Thí dụ: Sử dụng hàm strlen xác định độ dài một chuỗi nhập từ bàn phím.

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

main()
{
 char Chuoi[255];
 int Dodai ;
 printf("Nhap chuoi: ");gets(Chuoi);
 Dodai = strlen(Chuoi)
 printf("Chuoi vua nhap: ");puts(Chuoi);
 printf("Co do dai %d",Dodai);
 getch();
}
```

III.2.3 Đổi một ký tự thường thành ký tự hoa và ngược lại

Hàm toupper() (trong ctype.h) cho phép trả về ký tự hoa của 1 ký tự đầu vào là tham số của hàm. Ngược lại, hàm tolower() trả về ký tự thường của ký tự đầu vào là tham số của hàm.

Cú pháp: char toupper(char c)
char tolower(chart c)

III.2.4 Đổi chuỗi chữ thường thành chuỗi chữ hoa, hàm strupr()

Hàm struppr() được dùng để chuyển đổi chuỗi chữ thường thành chuỗi chữ hoa, kết quả trả về của hàm là một con trỏ chỉ đến địa chỉ chuỗi được chuyển đổi.

Cú pháp: char *strupr(char *s)

Thí dụ: Viết chương trình nhập vào một chuỗi ký tự từ bàn phím. Sau đó sử dụng hàm strupr() để chuyển đổi chúng thành chuỗi chữ hoa.

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

```
main(){
 char Chuoi[255],*s;
 printf("Nhap chuoi: ");gets(Chuoi);
 s=strupr(Chuoi) ;
 printf(“Chuoi chu hoa: ”);puts(s);
 getch();
}
```

III.2.5 Đổi chuỗi chữ hoa thành chuỗi chữ thường, hàm strlwr()

Muốn chuyển đổi chuỗi chữ hoa thành chuỗi toàn chữ thường, ta sử dụng hàm strlwr(), các tham số của hàm tương tự như hàm strupr()

Cú pháp: char *strlwr(char *s)

III.2.6 Sao chép chuỗi, hàm strcpy()

Hàm này được dùng để sao chép toàn bộ nội dung của chuỗi nguồn vào chuỗi đích.

Cú pháp: char *strcpy(char *Des, const char *Source)

Thí dụ: Viết chương trình cho phép chép toàn bộ chuỗi nguồn vào chuỗi đích.

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

main(){
 char Chuoi[255],s[255];
 printf("Nhap chuoi: ");gets(Chuoi);
 strcpy(s,Chuoi);
 printf(“Chuoi dich: ”);puts(s);
 getch();
}
```

III.2.7 Sao chép một phần chuỗi, hàm strncpy()

Hàm này cho phép chép n ký tự đầu tiên của chuỗi nguồn sang chuỗi đích.

Cú pháp:

char *strncpy(char *Des, const char *Source, size_t n)

Để ý rằng khi sao chép n ký tự đầu tiên của chuỗi nguồn vào chuỗi đích, ký tự '\0' không được tự động thêm vào chuỗi đích.

III.2.8 Trích một phần chuỗi, hàm strchr()

Để trích một chuỗi con của một chuỗi ký tự bắt đầu từ một ký tự được chỉ định trong chuỗi cho đến hết chuỗi, ta sử dụng hàm strchr().

Cú pháp : char *strchr(const char *str, int c)

Ghi chú:

- Nếu ký tự đã chỉ định không có trong chuỗi, kết quả trả về là NULL.

- Kết quả trả về của hàm là một con trỏ, con trỏ này chỉ đến ký tự c được tìm thấy đầu tiên trong chuỗi str.

III.2.9 Tìm kiếm nội dung chuỗi, hàm strstr()

Hàm strstr() được sử dụng để tìm kiếm sự xuất hiện đầu tiên của chuỗi s2 trong chuỗi s1.

Cú pháp: char *strstr(const char *s1, const char *s2)

Kết quả trả về của hàm là một con trỏ chỉ đến phần tử đầu tiên của chuỗi s1 có chứa chuỗi s2 hoặc giá trị NULL nếu chuỗi s2 không có trong chuỗi s1.

Thí dụ: Viết chương trình sử dụng hàm strstr() để lấy ra một phần của chuỗi gốc bắt đầu từ chuỗi "*hoc*".

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

main(){
 char Chuoi[255],*s;
 printf("Nhap chuoi: ");gets(Chuoi);
 s=strstr(Chuoi,"hoc");
 printf("Chuoi trich ra: ");puts(s);
 getch();
}
```

```
Nhap chuoi: Dai hoc Can Tho
Chuoi trich ra: hoc Can Tho
```

III.2.10 So sánh chuỗi, hàm strcmp()

Để so sánh hai chuỗi theo từng ký tự trong bảng mã Ascii, ta có thể sử dụng hàm strcmp().

Cú pháp: int strcmp(const char *s1, const char *s2)

Hai chuỗi s1 và s2 được so sánh với nhau, kết quả trả về là một số nguyên (số này có được bằng cách lấy ký tự của s1 trừ ký tự của s2 tại vị trí đầu tiên xảy ra sự khác nhau hoặc là 0 nếu 1 chuỗi giống nhau hoàn toàn).

- Nếu kết quả là số âm, chuỗi s1 nhỏ hơn chuỗi s2.
- Nếu kết quả là 0, hai chuỗi bằng nhau.
- Nếu kết quả là số dương, chuỗi s1 lớn hơn chuỗi s2.

III.2.11 So sánh chuỗi, hàm stricmp()

Hàm này thực hiện việc so sánh trong n ký tự đầu tiên của 2 chuỗi s1 và s2, giữa chữ thường và chữ hoa không phân biệt.

Cú pháp: int stricmp(const char *s1, const char *s2)

Kết quả trả về tương tự như kết quả trả về của hàm strcmp().

III.2.12 Khởi tạo chuỗi, hàm memset()

Hàm này được sử dụng để đặt n ký tự đầu tiên của chuỗi là ký tự c.

Cú pháp: memset(char *Des, int c, size_t n)

III.2.13 Đổi từ chuỗi ra số, hàm atoi(), atof(), atol() (trong stdlib.h)

Để chuyển đổi chuỗi ra số, ta sử dụng các hàm trên.

Cú pháp : int atoi(const char *s) : chuyển chuỗi thành số nguyên

long atol(const char *s) : chuyển chuỗi thành số nguyên dài

float atof(const char *s) : chuyển chuỗi thành số thực

Nếu chuyển đổi không thành công, kết quả trả về của các hàm là 0.

Ngoài ra, thư viện string.h còn hỗ trợ các hàm xử lý chuỗi khác, ta có thể đọc thêm trong phần trợ giúp.

IV. BÀI TẬP

1. Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình mã Ascii của từng ký tự có trong chuỗi.

2. Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình chuỗi đảo ngược của chuỗi đó. Thí dụ đảo của "abcd egh" là "hge dcba".

3. Viết chương trình nhập một chuỗi ký tự và kiểm tra xem chuỗi đó có đối xứng không.

Thí dụ : Chuỗi ABCDEDCBA là chuỗi đối xứng.

4. Nhập vào một chuỗi bất kỳ, hãy đếm số lần xuất hiện của mỗi ký tự có trong chuỗi.

5. Viết chương trình nhập vào một chuỗi.

- Hiển thị lên màn hình từ bên trái nhất và phần còn lại của chuỗi. Thí dụ: "Nguyen Van Minh" in ra thành:

Nguyen

Van Minh

- Hiển thị lên màn hình từ bên phải nhất và phần còn lại của chuỗi. Thí dụ: "Nguyen Van Minh" in ra thành:

Minh

Nguyen Van

6. Viết chương trình nhập vào một chuỗi rồi xuất chuỗi đó ra màn hình dưới dạng mỗi từ một dòng.

Thí dụ: "Nguyễn Văn Minh"

In ra:

Nguyen

Van

Minh

7. Viết chương trình nhập vào một chuỗi, in ra chuỗi đảo ngược của nó theo từng từ.

Thí dụ : chuỗi "Nguyen Van Minh" đảo thành "Minh Van Nguyen".

8. Viết chương trình nhập vào họ và tên của một người, cắt bỏ các khoảng trống không cần thiết (nếu có), tách tên ra khỏi họ và tên, in tên lên màn hình. Chú ý đến trường hợp cả họ và tên chỉ có một từ.

9. Viết chương trình nhập vào họ và tên của một người, cắt bỏ các khoảng trắng bên phải, trái và các khoảng trắng không có nghĩa trong chuỗi. In ra màn hình toàn bộ họ tên người đó dưới dạng chữ hoa, chữ thường.

10. Viết chương trình nhập vào một danh sách họ và tên của n người theo kiểu chữ thường, đổi các chữ cái đầu của họ, tên và chữ lót của mỗi người thành chữ hoa. In kết quả lên màn hình.

11. Viết chương trình nhập vào một danh sách họ và tên của n người, tách tên từng người ra khỏi họ và tên rồi sắp xếp danh sách tên theo thứ tự từ điển. In danh sách họ và tên sau khi đã sắp xếp.

Chương 9

KIỂU CẤU TRÚC - STRUCT

Chương này trình bày các vấn đề sau:

- *Khái niệm về kiểu cấu trúc.*
- *Cách sử dụng kiểu cấu trúc.*
- *Con trỏ cấu trúc.*

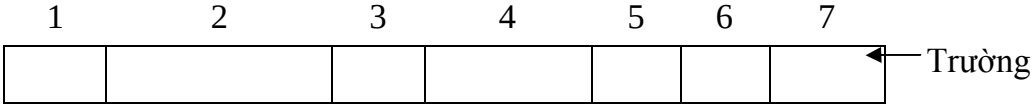
I. KIỂU CẤU TRÚC

I.1 Khái niệm

Kiểu cấu trúc (Structure) là kiểu dữ liệu bao gồm nhiều thành phần có kiểu khác nhau, mỗi thành phần được gọi là một trường (field).

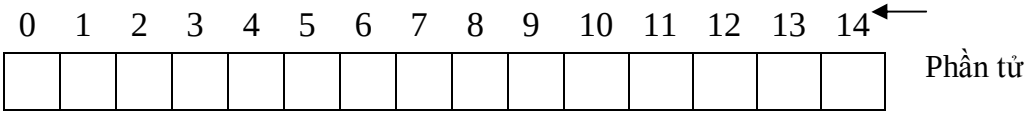
Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử của mảng là cùng kiểu còn các phần tử của kiểu cấu trúc có thể có kiểu khác nhau.

Hình ảnh của kiểu cấu trúc được minh họa:



Đây là cấu trúc có 7 trường

Còn kiểu mảng có dạng:



Đây là mảng có 15 phần tử

I.2 Định nghĩa kiểu cấu trúc

Cách 1:

struct <Tên cấu trúc>
{
    <Kiểu> <Trường 1>;
    <Kiểu> <Trường 2>;
    ……..
    <Kiểu> <Trường n>;
};

Cách 2: Sử dụng từ khóa **typedef** để định nghĩa kiểu:

typedef struct
{
    <Kiểu> <Trường 1>;
    <Kiểu> <Trường 2>;
    ……..
    <Kiểu> <Trường n>;
} <Tên cấu trúc>;

Trong đó:

- <Tên cấu trúc>: là một tên được đặt theo quy tắc đặt tên của danh biểu; tên này mang ý nghĩa sẽ là tên kiểu cấu trúc.

- <Kiểu> <Trường i> (i=1..n): mỗi trường trong cấu trúc có dữ liệu thuộc kiểu gì (tên của trường phải là một tên được đặt theo quy tắc đặt tên của danh biểu).

Thí dụ 1: Để quản lý ngày, tháng, năm của một ngày trong năm ta có thể khai báo kiểu cấu trúc gồm 3 thông tin: ngày, tháng, năm.

```
struct NgayThang{
 unsigned char Ngay;
 unsigned char Thang;
 unsigned int Nam;
};
```

```
typedef struct{
  unsigned char Ngay;
  unsigned char Thang;
  unsigned int Nam;
}NgayThang;
```

Thí dụ 2: Mỗi sinh viên cần được quản lý bởi các thông tin: mã số sinh viên, họ tên, ngày tháng năm sinh, giới tính, địa chỉ thường trú. Lúc này ta có thể khai báo một struct gồm các thông tin trên.

```
struct SinhVien{
 char MSSV[10];
 char HoTen[40];
 struct NgayThang NgaySinh;
 int Phai;
 char DiaChi[40];
};
```

```
typedef struct{
  char MSSV[10];
  char HoTen[40];
  NgayThang NgaySinh;
  int Phai;
  char DiaChi[40];
} SinhVien;
```

I.3 Khai báo biến cấu trúc

Việc khai báo biến cấu trúc cũng tương tự như khai báo biến thuộc kiểu dữ liệu chuẩn.

Cú pháp:

- Đối với cấu trúc được định nghĩa theo cách 1:

struct <Tên cấu trúc> <Biến 1> [, <Biến 2>…];

- Đối với các cấu trúc được định nghĩa theo cách 2:

<Tên cấu trúc> <Biến 1> [, <Biến 2>…];

Thí dụ: Khai báo biến NgaySinh có kiểu cấu trúc NgayThang; biến SV có kiểu cấu trúc SinhVien.

```
struct NgayThang NgaySinh;
struct SinhVien SV;
```

```
NgayThang NgaySinh;
SinhVien SV;
```

II. CÁC THAO TÁC TRÊN BIẾN KIỂU CẤU TRÚC

II.1 Truy xuất đến từng trường của biến cấu trúc

Cú pháp: <Biến cấu trúc>.<Tên trường>

Khi sử dụng cách truy xuất theo kiểu này, các thao tác trên <Biến cấu trúc>.<Tên trường> giống như các thao tác trên các biến của kiểu dữ liệu của <Tên trường>.

Thí dụ 1: Viết chương trình cho phép đọc dữ liệu từ bàn phím cho biến cấu trúc SinhVien và hiển thị biến cấu trúc đó lên màn hình:

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

typedef struct{
   unsigned char Ngay;
   unsigned char Thang;
   unsigned int Nam;
} NgayThang;

typedef struct{
   char MSSV[10];
   char HoTen[40];
   NgayThang NgaySinh;
   int Phai;
   char DiaChi[40];
} SinhVien;

   // Hàm in lên màn hình 1 mẫu tin SinhVien
void InSV(SinhVien s){
   printf("MSSV:  |  Ho va ten   |  Ngay Sinh   |
                   Dia chi\n");
   printf("%s  |  %s |  %d-%d-%d |  %s\n",
       s.MSSV,s.HoTen,s.NgaySinh.Ngay,
       s.NgaySinh.Thang,s.NgaySinh.Nam,s.DiaChi);
}

main(){
   SinhVien SV,s;
   printf("Nhap MSSV: ");gets(SV.MSSV);
   printf("Nhap Ho va ten: ");gets(SV.HoTen);
   printf("Sinh ngay: ");
   scanf("%d",&SV.NgaySinh.Ngay);
```

```
    printf("Thang: ");
    scanf("%d",&SV.NgaySinh.Thang);
    printf("Nam: ");scanf("%d",&SV.NgaySinh.Nam);
    printf("Gioi tinh (0: Nu), (1: Nam): ");
    scanf("%d",&SV.Phai);
    fflush(stdin);
    printf("Dia chi: ");gets(SV.DiaChi);
    InSV(SV);
    s=SV;      // Gán trị cho cấu trúc s
    InSV(s);
    getch();
}
```

```
Nhap MSSV: 1040393
Nhap Ho va ten: Lam Nhat Tien
Sinh ngay: 29
Thang: 8
Nam: 1986
Gioi tinh (0: Nu), (1: Nam): 1
Dia chi: 1 Ly Tu Trong
MSSV:    |       Ho va ten       |       Ngay Sinh       |       Dia chi
1040393 |       Lam Nhat Tien   |       29-8-1986       |       1 Ly Tu Trong
MSSV:    |       Ho va ten       |       Ngay Sinh       |       Dia chi
1040393 |       Lam Nhat Tien   |       29-8-1986       |       1 Ly Tu Trong
```

Lưu ý:

- Các biến cấu trúc có thể gán cho nhau. Thực chất đây là thao tác trên toàn bộ cấu trúc không phải trên một trường riêng rẽ nào. Chương trình trên dòng s=SV là một ví dụ.

- Với các biến kiểu cấu trúc ta không thể thực hiện được các thao tác sau đây:

  - Sử dụng các hàm xuất nhập trên biến cấu trúc.
  - Các phép toán quan hệ, các phép toán số học và logic.

Thí dụ 2: Nhập vào hai số phức và tính tổng của chúng. Ta biết rằng số phức là một cặp (a,b) trong đó a, b là các số thực, a gọi là phần thực, b là phần ảo. (Đôi khi người ta cũng viết số phức dưới dạng a + ib trong đó i là một đơn vị ảo có tính chất i^2=-1). Gọi số phức c1=(a1, b1) và c2=(a2,b2) khi đó tổng của hai số phức c1 và c2 là một số phức c3 mà c3=(a1+a2, b1+b2). Với hiểu biết như vậy ta có thể xem mỗi số phức là một cấu trúc có hai trường, một trường biểu diễn cho phần thực, một trường biểu diễn cho phần ảo. Việc tính tổng của hai số phức được tính bằng cách lấy phần thực cộng với phần thực và phần ảo cộng với phần ảo.

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

typedef struct{
      float Thuc;
      float Ao;
} SoPhuc;
      // Hàm in số phức lên màn hình
void InSoPhuc(SoPhuc p){
      printf("%.2f + i%.2f\n",p.Thuc,p.Ao);
}

main(){
      SoPhuc p1,p2,p;
      printf("Nhap so phuc thu nhat:\n");
      printf("Phan thuc: ");
      scanf("%f",&p1.Thuc);
      printf("Phan ao: ");scanf("%f",&p1.Ao);
      printf("Nhap so phuc thu hai:\n");
      printf("Phan thuc: ");
      scanf("%f",&p2.Thuc);
      printf("Phan ao: ");scanf("%f",&p2.Ao);
      printf("So phuc thu nhat: ");
      InSoPhuc(p1);
      printf("So phuc thu hai: ");
      InSoPhuc(p2);
      p.Thuc = p1.Thuc+p2.Thuc;
      p.Ao = p1.Ao + p2.Ao;
      printf("Tong 2 so phuc: ");
      InSoPhuc(p);
      getch();
}
```

Kết quả thực hiện chương trình:

```
Nhap so phuc thu nhat:
Phan thuc: 3
Phan ao: 4
Nhap so phuc thu hai:
Phan thuc: 8
Phan ao: 9
So phuc thu nhat: 3.00 + i4.00
So phuc thu hai: 8.00 + i9.00
Tong 2 so phuc: 11.00 + i13.00
```

II.2 Khởi tạo cấu trúc

Việc khởi tạo cấu trúc có thể được thực hiện trong lúc khai báo biến cấu trúc. Các trường của cấu trúc được khởi tạo được đặt giữa 2 dấu { và }, chúng được phân cách nhau bởi dấu phẩy (,).

Thí dụ: Khởi tạo biến cấu trúc NgaySinh:

```
struct NgayThang NgaySinh ={29, 8, 1986};
```

III. CON TRỎ CẤU TRÚC

III.1 Khai báo

Việc khai báo một biến con trỏ kiểu cấu trúc cũng tương tự như khi khai báo một biến con trỏ khác, nghĩa là đặt thêm dấu * vào phía trước tên biến.

Cú pháp: struct <Tên cấu trúc> * <Tên biến con trỏ>;

Thí dụ: Ta có thể khai báo một con trỏ cấu trúc kiểu NgayThang như sau:

```
struct NgayThang *p;
hay NgayThang *p;  // Nếu có định nghĩa kiểu
```

III.2 Sử dụng các con trỏ kiểu cấu trúc

Khi khai báo biến con trỏ cấu trúc, biến con trỏ chưa chỉ đến 1 địa chỉ cụ thể. Lúc này nó chỉ mới được cấp phát 2 byte để lưu giữ địa chỉ và được ghi nhận là con trỏ chỉ đến 1 cấu trúc, nhưng chưa chỉ đến 1 đối tượng rõ ràng. Muốn thao tác trên con trỏ cấu trúc hợp lệ, cũng tương tự như các con trỏ khác, ta phải:

- Cấp phát một vùng nhớ cho nó (sử dụng hàm malloc() hay calloc())

- Hoặc, cho nó quản lý địa chỉ của một biến cấu trúc nào đó.

Thí dụ: Sau khi khởi tạo giá trị của cấu trúc:

```
struct NgayThang Ngay = {29,8,1986};
p = &Ngay;
```

lúc này biến con trỏ p đã chứa địa chỉ của Ngay.

III.2 Truy cập các thành phần của cấu trúc đang được quản lý bởi con trỏ

Để truy cập đến từng trường của 1 cấu trúc thông qua con trỏ của nó, ta sử dụng toán tử dấu mũi tên (->: dấu - và dấu >).

Ngoài ra, ta vẫn có thể sử dụng đến phép toán * để truy cập vùng dữ liệu đang được quản lý bởi con trỏ cấu trúc để lấy thông tin cần thiết.

Thí dụ: Sử dụng con trỏ cấu trúc.

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

typedef struct{
   unsigned char Ngay;
   unsigned char Thang;
   unsigned int Nam;
} NgayThang;

main(){
   NgayThang Ng={29,8,1986};
   NgayThang *p;
   p=&Ng;
   printf("Truy cap binh thuong %d-%d-%d\n",
         Ng.Ngay,Ng.Thang,Ng.Nam);
   printf("Truy cap qua con tro %d-%d-%d\n",
         p->Ngay,p->Thang,p->Nam);
   printf("Truy cap qua vung nho con tro
      %d-%d-%d\n",(*p).Ngay,(*p).Thang,(*p).Nam);
   getch();
}
```

Kết quả:

```
Truy cap binh thuong 29-8-1986
Truy cap qua con tro 29-8-1986
Truy cap qua vung nho con tro 29-8-1986
```

IV. BÀI TẬP

1. Hãy định nghĩa kiểu:

```
struct Hoso{
      char HoTen[40];
      float Diem;
      char Loai[10];
};
```

Viết chương trình nhập vào họ tên, điểm của n học sinh. Xếp loại văn hóa theo cách sau:

Điểm	Xếp loại
>=9.0	Xuat sac
8.0 -< 9.0	Gioi
7.0 - <8.0	Khá
5.0 - <7.0	Trung bình
<5.0	Không đạt

Hiển thị danh sách lên màn hình theo dạng sau (thứ tự giảm dần theo điểm trung bình):

XEP LOAI VAN HOA

HO VA TEN	DIEM	XEP LOAI
Nguyen Van A	7.0	Kha
Ho Thi B	5.0	Trung binh
Dang Kim C	4.0	Khong dat

.....

2. Xem một phân số là một cấu trúc có hai trường là tử số và mẫu số. Hãy viết chương trình thực hiện các phép toán cộng, trừ, nhân, chia hai phân số. (Các kết quả phải tối giản).

3. Tạo một danh sách cán bộ công nhân viên, mỗi người người xem như một cấu trúc bao gồm các trường Ho, Ten, Luong, Tuoi, Diachi. Nhập một số người vào danh sách, sắp xếp tên theo thứ tự từ điển, in danh sách đã sắp xếp theo mẫu sau:

DANH SACH CAN BO CONG NHAN VIEN

STT	HO VA TEN	LUONG	TUOI	DIACHI
1	Nguyen Van A	333.00	26	Can Tho
2	Dang Kim B	290.00	23	Vinh Long

Chương 10

TẬP TIN

Các vấn đề được trình bày trong chương này:

- *Các khái niệm liên quan đến tập tin.*
- *Các bước thao tác với tập tin.*
- *Một số hàm truy xuất tập tin văn bản.*
- *Một số hàm truy xuất tập tin nhị phân.*

I. MỘT SỐ KHÁI NIỆM VỀ TẬP TIN

Đối với các kiểu dữ liệu ta đã biết như kiểu số, kiểu mảng, kiểu cấu trúc thì dữ liệu được tổ chức trong bộ nhớ trong (RAM) của máy tính nên khi kết thúc việc thực hiện chương trình thì dữ liệu cũng bị mất; khi cần chúng ta bắt buộc phải nhập lại từ bàn phím. Điều đó vừa mất thời gian vừa không giải quyết được các bài toán với số liệu lớn. Để giải quyết vấn đề, người ta đưa ra kiểu tập tin (file) cho phép lưu trữ dữ liệu ở bộ nhớ ngoài (đĩa). Khi kết thúc chương trình thì dữ liệu vẫn còn do đó chúng ta có thể sử dụng nhiều lần. Một đặc điểm khác của kiểu tập tin là kích thước lớn với số lượng các phần tử không hạn chế (chỉ bị hạn chế bởi dung lượng của bộ nhớ ngoài).

Có 2 loại dữ liệu kiểu tập tin:

- Tập tin văn bản (Text File): là loại tập tin dùng để ghi các ký tự lên đĩa. Điểm đặc biệt là dữ liệu của tập tin được lưu trữ thành các dòng, mỗi dòng được kết thúc bằng ký tự xuống dòng (new line), ký hiệu '\n'; ký tự này là sự kết hợp của 2 ký tự CR (Carriage Return - Về đầu dòng, mã Ascii là 13) và LF (Line Feed - Xuống dòng, mã Ascii là 10). Mỗi tập tin được kết thúc bởi ký tự EOF (End Of File) có mã Ascii là 26 (xác định bởi tổ hợp phím Ctrl + Z).

Tập tin văn bản chỉ có thể truy xuất theo kiểu tuần tự.

- Tập tin nhị phân: là loại tập tin chứa 1 dãy liên tục các byte (mã Ascii của các ký tự). Tập tin nhị phân có 2 loại:

  - Tập tin định kiểu (Typed File): là loại tập tin bao gồm nhiều phần tử có cùng kiểu: char, int, long, cấu trúc… và được lưu trữ trên đĩa dưới dạng một chuỗi các byte liên tục.

  - Tập tin không định kiểu (Untyped File): là loại tập tin mà dữ liệu của chúng gồm các cấu trúc dữ liệu mà người ta không quan tâm đến nội dung hoặc kiểu của nó, chỉ lưu ý đến các yếu tố vật lý của tập tin như độ lớn và các yếu tố tác động lên tập tin mà thôi.

Biến tập tin: là một biến thuộc kiểu dữ liệu tập tin dùng để đại diện cho một tập tin. Dữ liệu chứa trong một tập tin được truy xuất qua các thao tác với thông số là biến tập tin đại diện cho tập tin đó.

Con trỏ tập tin: Khi một tập tin được mở ra để làm việc, tại mỗi thời điểm, sẽ có một vị trí của tập tin mà tại đó việc đọc/ghi thông tin sẽ xảy ra. Người ta hình dung có một con trỏ đang chỉ đến vị trí đó và đặt tên nó là con trỏ tập tin.

Sau khi đọc/ghi xong dữ liệu, con trỏ sẽ chuyển dịch thêm một phần tử về phía cuối tập tin. Sau phần tử dữ liệu cuối cùng của tập tin là dấu kết thúc tập tin EOF (End Of File).

II. CÁC THAO TÁC TRÊN TẬP TIN

Muốn thao tác trên tập tin, ta phải lần lượt làm theo các bước:

- Khai báo biến tập tin.
- Mở tập tin bằng hàm fopen().
- Thực hiện các thao tác xử lý dữ liệu của tập tin bằng các hàm đọc/ghi dữ liệu.
- Đóng tập tin bằng hàm fclose().

Lưu ý: Các thao tác liên quan đến tập tin sử dụng các hàm trong thư viện stdio.h.

II.1. Khai báo biến tập tin

Cú pháp: FILE <Danh sách các biến con trỏ>

Các biến trong danh sách phải là các con trỏ và được phân cách bởi dấu phẩy(,).

Thí dụ: FILE *f1**,***f2;

II.2. Mở tập tin

Cú pháp: FILE *fopen(char *Path, const char *Mode)

Trong đó:

- Path: chuỗi chỉ đường dẫn đến tập tin trên đĩa.

- Type: chuỗi xác định cách thức mà tập tin sẽ mở. Các giá trị có thể của *Mode*:

Chế độ	Ý nghĩa
r	Mở tập tin văn bản để đọc
w	Tạo ra tập tin văn bản mới để ghi
a	Nối vào tập tin văn bản
rb	Mở tập tin nhị phân để đọc
wb	Tạo ra tập tin nhị phân để ghi
ab	Nối vào tập tin nhị phân
r+	Mở một tập tin văn bản để đọc/ghi
w+	Tạo ra tập tin văn bản để đọc ghi
a+	Nối vào hay tạo mới tập tin văn bản để đọc/ghi
r+b	Mở ra tập tin nhị phân để đọc/ghi
w+b	Tạo ra tập tin nhị phân để đọc/ghi
a+b	Nối vào hay tạo mới tập tin nhị phân

- Hàm fopen trả về một con trỏ tập tin. Chương trình của ta không thể thay đổi giá trị của con trỏ này. Nếu có một lỗi xuất hiện trong khi mở tập tin thì hàm này trả về con trỏ NULL.

Thí dụ: Mở một tập tin tên TEST.txt để ghi.

```
FILE *f;
f = fopen("TEST.txt", "w");
if (f!=NULL)
{
  // Các câu lệnh để thao tác với tập tin
  // Đóng tập tin
}
```

Trong thí dụ trên, ta có sử dụng câu lệnh kiểm tra điều kiện để xác định mở tập tin có thành công hay không?.

Nếu mở tập tin để ghi, nếu tập tin đã tồn tại rồi thì tập tin sẽ bị xóa và một tập tin mới được tạo ra. Nếu ta muốn ghi nối dữ liệu, ta phải sử dụng chế độ "a". Khi mở với chế độ đọc, tập tin phải tồn tại rồi, nếu không một lỗi sẽ xuất hiện.

II.3. Đóng tập tin

Hàm fclose() được dùng để đóng tập tin được mở bởi hàm fopen(). Hàm này sẽ ghi dữ liệu còn lại trong vùng đệm vào tập tin và đóng lại tập tin.

Cú pháp: int fclose(FILE *f)

Trong đó f là con trỏ tập tin được mở bởi hàm fopen(). Giá trị trả về của hàm là 0 báo rằng việc đóng tập tin thành công. Hàm trả về EOF nếu có xuất hiện lỗi.

II.4. Kiểm tra đến cuối tập tin hay chưa?

Cú pháp: int feof(FILE *f)

Ý nghĩa: Kiểm tra xem đã chạm tới cuối tập tin hay chưa và trả về EOF nếu cuối tập tin được chạm tới, ngược lại trả về 0.

II.5 Di chuyển con trỏ tập tin về đầu tập tin - Hàm rewind()

Khi ta đang thao tác một tập tin đang mở, con trỏ tập tin luôn di chuyển về phía cuối tập tin. Muốn cho con trỏ quay về đầu tập tin như khi mở nó, ta sử dụng hàm *rewind()*.

Cú pháp: void rewind(FILE *f)

III. TRUY CẬP TẬP TIN VĂN BẢN

III.1. Ghi dữ liệu lên tập tin văn bản

III.1.1 Hàm putc(): Hàm này được dùng để ghi một ký tự lên một tập tin văn bản đang được mở để làm việc.

Cú pháp: int putc(int c, FILE *f)

Trong đó, tham số c chứa mã Ascii của một ký tự nào đó. Mã này được ghi lên tập tin liên kết với con trỏ f. Hàm này trả về EOF nếu gặp lỗi.

III.1.2 Hàm fputs(): Hàm này dùng để ghi một chuỗi ký tự chứa trong vùng đệm lên tập tin văn bản.

Cú pháp: int puts(const char *buffer, FILE *f)

Trong đó, buffer là con trỏ có kiểu char chỉ đến vị trí đầu tiên của chuỗi ký tự được ghi vào. Hàm này trả về giá trị 0 nếu buffer chứa chuỗi rỗng và trả về EOF nếu gặp lỗi.

III.1.3 Hàm fprintf(): Hàm này dùng để ghi dữ liệu có định dạng lên tập tin văn bản.

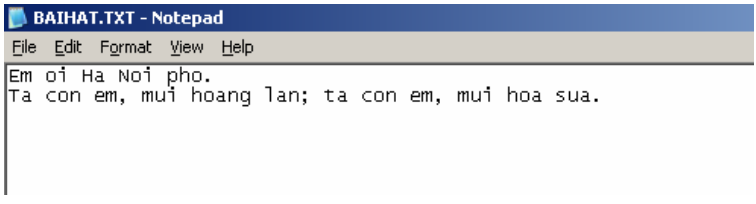
Cú pháp: void fprintf(FILE *f, const char *format, varexpr)

Trong đó: format: chuỗi định dạng (giống với các định dạng của hàm printf()), varexpr: danh sách các biểu thức, mỗi biểu thức cách nhau dấu phẩy (,).

Thí dụ: Viết chương trình ghi chuỗi ký tự lên tập tin văn bản D:\\Baihat.txt

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

main(){
   FILE *f;
   f=fopen("D:\\Baihat.txt","r+");
   if (f!=NULL)
   {
     fputs("Em oi Ha Noi pho.\n",f);
     fputs("Ta con em, mui hoang lan; ta con em,
              mui hoa sua.",f);
     fclose(f);
   }
   getch();
}
```

Nội dung tập tin Baihat.txt khi được mở bằng trình soạn thảo văn bản Notepad.



III.2. Đọc dữ liệu từ tập tin văn bản

III.2.1 Hàm getc() : Hàm này dùng để đọc dữ liệu từ tập tin văn bản đang được mở để làm việc.

Cú pháp: int getc(FILE *f)

Hàm này trả về mã Ascii của một ký tự nào đó (kể cả EOF) trong tập tin liên kết với con trỏ f.

III.2.2 Hàm fgets()

Cú pháp: char *fgets(char *buffer, int n, FILE *f)

Hàm này được dùng để đọc một chuỗi ký tự từ tập tin văn bản đang được mở ra và liên kết với con trỏ f cho đến khi đọc đủ n ký tự hoặc gặp ký tự xuống dòng '\n' (ký tự này cũng được đưa vào chuỗi kết quả) hay gặp ký tự kết thúc EOF (ký tự này không được đưa vào chuỗi kết quả).

Trong đó:

- buffer (vùng đệm): con trỏ có kiểu char chỉ đến cùng nhớ đủ lớn chứa các ký tự nhận được.

- n: giá trị nguyên chỉ độ dài lớn nhất của chuỗi ký tự nhận được.

- f: con trỏ liên kết với một tập tin nào đó.

- Ký tự NULL ('\0') tự động được thêm vào cuối chuỗi kết quả lưu trong vùng đêm.

- Hàm trả về địa chỉ đầu tiên của vùng đệm khi không gặp lỗi và chưa gặp ký tự kết thúc EOF. Ngược lại, hàm trả về giá trị NULL.

III.2.3 Hàm fscanf(): Hàm này dùng để đọc dữ liệu từ tập tin văn bản vào danh sách các biến theo định dạng.

Cú pháp: void fscanf(FILE *f, const char *format, pointers)

Trong đó: format: chuỗi định dạng (giống hàm scanf()); pointers: danh sách địa chỉ các biến mỗi biến cách nhau dấu phẩy (,).

Thí dụ: Viết chương trình chép tập tin D:\Baihat.txt ở trên sang tập tin D:\Baica.txt.

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

main(){
 FILE *f1,*f2;
 f1=fopen("D:\\Baihat.txt","rt");
 f2=fopen("D:\\Baica.txt","wt");
 if (f1!=NULL && f2!=NULL)
 {
      int ch=fgetc(f1);
      while (! feof(f1))
      {
           fputc(ch,f2);
           ch=fgetc(f1);
      }
      fclose(f1);
      fclose(f2);
 }
 getch();
}
```

IV. TRUY CẬP TẬP TIN NHỊ PHÂN

IV.1 Ghi dữ liệu lên tập tin nhị phân - Hàm fwrite()

Cú pháp:

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

Trong đó:

- ptr: con trỏ chỉ đến vùng nhớ chứa thông tin cần ghi lên tập tin.
- n: số phần tử sẽ ghi lên tập tin.
- size: kích thước của mỗi phần tử.
- f: con trỏ tập tin đã được mở.
- Giá trị trả về của hàm này là số phần tử được ghi lên tập tin. Giá trị này bằng n trừ khi xuất hiện lỗi.

IV.2 Đọc dữ liệu từ tập tin nhị phân - Hàm fread()

Cú pháp:

size_t fread(const void *ptr, size_t size, size_t n, FILE *f)

Trong đó:

- ptr: con trỏ chỉ đến vùng nhớ sẽ nhận dữ liệu từ tập tin.
- n: số phần tử được đọc từ tập tin.
- size: kích thước của mỗi phần tử.
- f: con trỏ tập tin đã được mở.
- Giá trị trả về của hàm này là số phần tử đã đọc được từ tập tin. Giá trị này bằng n hay nhỏ hơn n nếu đã chạm đến cuối tập tin hoặc có lỗi xuất hiện..

IV.3 Di chuyển con trỏ tập tin - Hàm fseek()

Việc ghi hay đọc dữ liệu từ tập tin sẽ làm cho con trỏ tập tin dịch chuyển một số byte, đây chính là kích thước của kiểu dữ liệu của mỗi phần tử của tập tin.

Khi đóng tập tin rồi mở lại nó, con trỏ luôn ở vị trí ngay đầu tập tin. Nhưng nếu ta sử dụng kiểu mở tập tin là "a" để ghi nối dữ liệu, con trỏ tập tin sẽ di chuyển đến vị trí cuối cùng của tập tin này.

Ta cũng có thể điều khiển việc di chuyển con trỏ tập tin đến vị trí chỉ định bằng hàm fseek().

Cú pháp: int fseek(FILE *f, long offset, int whence)

Trong đó:

- f: con trỏ tập tin đang thao tác.
- offset: số byte cần dịch chuyển con trỏ tập tin kể từ vị trí trước đó. Phần tử đầu tiên là vị trí 0.
- whence: vị trí bắt đầu để tính offset, ta có thể chọn điểm xuất phát là:

0	**SEEK_SET**	Vị trí đầu tập tin
1	**SEEK_CUR**	Vị trí hiện tại của con trỏ tập tin
2	**SEEK_END**	Vị trí cuối tập tin

- Kết quả trả về của hàm là 0 nếu việc di chuyển thành công. Nếu không thành công, 1 giá trị khác 0 (đó là 1 mã lỗi) được trả về.

IV.3 Thí dụ

Thí dụ 1: Viết chương trình ghi lên tập tin CacSo.Dat 3 giá trị số (thực, nguyên, nguyên dài). Sau đó đọc các số từ tập tin vừa ghi và hiển thị lên màn hình.

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

main(){
   FILE *f;
   f=fopen("D:\\CacSo.txt","wb");
   if (f!=NULL)
   {
         double d=3.14;
         int i=101;
         long l=54321;
         fwrite(&d,sizeof(double),1,f);
         fwrite(&i,sizeof(int),1,f);
         fwrite(&l,sizeof(long),1,f);
         // Doc tu tap tin
         rewind(f);
         fread(&d,sizeof(double),1,f);
         fread(&i,sizeof(int),1,f);
         fread(&l,sizeof(long),1,f);
         printf("Cac ket qua la: %f %d   %ld",
                     d,i,l);
         fclose(f);
   }
   getch();
}
```

Thí dụ 2: Mỗi sinh viên cần quản lý ít nhất 2 thông tin: mã sinh viên và họ tên. Viết chương trình cho phép lựa chọn các chức năng: nhập danh sách sinh viên từ bàn phím rồi ghi lên tập tin SinhVien.dat, đọc dữ liệu từ tập tin SinhVien.dat rồi hiển thị danh sách lên màn hình, tìm kiếm họ tên của một sinh viên nào đó dựa vào mã sinh viên nhập từ bàn phím.

Ta nhận thấy rằng mỗi phần tử của tập tin SinhVien.Dat là một cấu trúc có 2 trường: mã và họ tên. Do đó, ta cần khai báo cấu trúc này và sử dụng các hàm đọc/ghi tập tin nhị phân với kích thước mỗi phần tử của tập tin là chính kích thước cấu trúc đó.

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

typedef struct{
   char Ma[10];
   char HoTen[40];
} SinhVien;

void WriteFile(char *FileName){
   FILE *f;
   int n, i;
   SinhVien sv;
   f=fopen(FileName,"ab");
   if (f==NULL) return;
   printf("Nhap bao nhieu sinh vien? ");
   scanf("%d",&n);
   fflush(stdin);
   for(i=1;i<=n;i++){
         printf("Sinh vien thu %i\n",i);
         printf("  - MSSV: ");gets(sv.Ma);
         printf("  - Ho ten: ");gets(sv.HoTen);
         fwrite(&sv,sizeof(sv),1,f);
         fflush(stdin);
   }
   fclose(f);
   printf("Bam phim bat ky de tiep tuc");
   getch();
}

void ReadFile(char *FileName){
   FILE *f;
   SinhVien sv;
   f=fopen(FileName,"rb");
   if (f==NULL) return;
   printf("     MSSV     |     Ho va ten\n");
   fread(&sv,sizeof(sv),1,f);
   while (!feof(f)){
         printf("   %s    |  %s\n",
                     sv.Ma,sv.HoTen);
         fread(&sv,sizeof(sv),1,f);
   }
   fclose(f);
   printf("Bam phim bat ky de tiep tuc!!!");
   getch();
}
```

```
void Search(char *FileName){
   char MSSV[10];
   FILE *f;
   int Found;
   SinhVien sv;
   fflush(stdin);
   printf("Ma so sinh vien can tim: ");
   gets(MSSV);
   Found=0;
   f=fopen(FileName,"rb");
   if (f==NULL) return;
   while (!feof(f) && Found==0){
        fread(&sv,sizeof(sv),1,f);
        if (strcmp(sv.Ma,MSSV)==0) Found=1;
   }
   fclose(f);

   if (Found == 1)
    printf("Tim thay SV co ma %s. Ho ten la: %s",
        sv.Ma,sv.HoTen);
   else
    printf("Tim khong thay sinh vien co ma %s",
        MSSV);

   printf("\nBam phim bat ky de tiep tuc!!!");
   getch();
}

main(){
   int c;
   for (;;){
        printf("1. Nhap DSSV\n");
        printf("2. In DSSV\n");
        printf("3. Tim kiem\n");
        printf("4. Thoat\n");
        printf("Ban chon 1, 2, 3, 4: ");
        scanf("%d",&c);
        if(c==1)
             WriteFile("d:\\SinhVien.Dat");
        else if (c==2)
             ReadFile("d:\\SinhVien.Dat");
        else if (c==3)
             Search("d:\\SinhVien.Dat");
        else break;
   }
}
```

V. BÀI TẬP

1. Viết chương trình quản lý một tập tin văn bản theo các yêu cầu:

- Nhập từ bàn phím nội dung một văn bản sau đó ghi vào đĩa.

- Đọc từ đĩa nội dung văn bản vừa nhập và in lên màn hình.

- Đọc từ đĩa nội dung văn bản vừa nhập, in nội dung đó lên màn hình và cho phép nối thêm thông tin vào cuối tập tin đó.

2. Viết chương trình cho phép thống kê số lần xuất hiện của các ký tự là chữ ('A'..'Z','a'..'z') trong một tập tin văn bản.

3. Viết chương trình đếm số từ và số dòng trong một tập tin văn bản.

4. Viết chương trình nhập từ bàn phím và ghi vào 1 tập tin tên là DMHH.DAT với mỗi phần tử của tập tin là 1 cấu trúc bao gồm các trường: Ma (mã hàng :char[5]), Ten (Tên hàng: char[20]).Kết thúc việc nhập bằng cách gõ ENTER vào Ma. Ta sẽ dùng tập tin này để giải mã hàng hóa cho tập tin DSHH.DAT sẽ đề cập trong bài 5.

5. Viết chương trình cho phép nhập từ bàn phím và ghi vào 1 tập tin tên DSHH.Dat với mỗi phần tử của tập tin là một cấu trúc bao gồm các trường : mh (mã hàng: char[5]), sl (số lượng : int), dg (đơn giá: float), st (Số tiền: float) theo yêu cầu:

- Mỗi lần nhập một cấu trúc

- Trước tiên nhập mã hàng (mh), đưa mh so sánh với Ma trong tập tin DMHH.DAT đã được tạo ra bởi bài tập 4, nếu mh=ma thì in tên hàng ngay bên cạnh mã hàng.

- Nhập số lượng (sl).

- Nhập đơn giá (dg).

- Tính số tiền = số lượng * đơn giá.

Kết thúc việc nhập bằng cách đánh ENTER vào mã hàng. Sau khi nhập xong yêu cầu in toàn bộ danh sách hàng hóa có sự giải mã về tên hàng theo mẫu sau:

STT	MA HANG	TEN HANG	SOLG	DON GIA	SO TIEN
1	a0101	Duong cat trang	25	10000.00	250000.00
2	b0101	Sua co gai Ha Lan	10	40000.00	400000.00

TÀI LIỆU THAM KHẢO

[1] Nguyễn Văn Linh, *Giáo trình Tin Học Đại Cương A*, Khoa Công Nghệ Thông Tin, Đại học Cần Thơ, 1991.

[2] Nguyễn Đình Tê, Hoàng Đức Hải , *Giáo trình lý thuyết và bài tập ngôn ngữ C*; Nhà xuất bản Giáo dục, 1999.

[3] Nguyễn Cẩn, *C – Tham khảo toàn diện*, Nhà xuất bản Đồng Nai, 1996.

[4] Võ Văn Viện, *Giúp tự học Lập Trình với ngôn ngữ C*, Nhà xuất bản Đồng Nai, 2002.

[5] Brain W. Kernighan & Dennis Ritchie, *The C Programming Language*, Prentice Hall Publisher, 1988.

[6] Trần Đức Huyên, *Phương pháp giải các bài toán trong Tin học*, Nhà xuất bản Giáo dục, 2003.