

HỌC VIỆN KỸ THUẬT QUÂN SỰ
KHOA CÔNG NGHỆ THÔNG TIN

Bài giảng
KỸ THUẬT LẬP TRÌNH

Biên soạn
Hà Đại Dương
Nguyễn Mậu Uyên

Hà Nội - 1/2013

MỤC LỤC

Bài 1 - TỔNG QUAN.....	4
I. Giới thiệu.....	4
II. Bài toán và việc giải bài toán trên máy tính.....	5
III. Công cụ lập trình.....	10
IV. Tóm tắt nội dung bài học.....	13
V. Bài tập.....	14
Bài 2 - NGÔN NGỮ LẬP TRÌNH C/C++.....	15
I. Giới thiệu.....	15
II. Một số khái niệm cơ bản.....	16
III. Cấu trúc một chương trình đơn giản.....	25
IV. Nhập/Xuất dữ liệu.....	27
V. Tóm tắt nội dung bài học.....	32
VI. Bài tập.....	33
Bài 3 - Bài thực hành: MÔI TRƯỜNG LẬP TRÌNH VÀ CÁC LỆNH VÀO/RA.....	34
I. Làm quen môi trường Dev-C++.....	34
II. Bài tập làm theo yêu cầu.....	34
III. Bài tập tự làm.....	37
Bài 4 - ĐIỀU KHIỂN CHỌN.....	40
I. Khỏi lệnh.....	40
II. Lệnh IF.....	41
III. Lệnh SWITCH.....	44
IV. Tóm tắt.....	48
V. Bài tập.....	48
Bài 5 - Bài thực hành: ĐIỀU KHIỂN CHỌN.....	49
I. Bài tập làm theo yêu cầu.....	49
II. Bài tập tự làm.....	52
Bài 6 - ĐIỀU KHIỂN LẶP.....	53
I. Lệnh FOR.....	53
II. Lệnh WHILE.....	55
III. Lệnh DO .. WHILE.....	57
IV. Lệnh break và continue.....	60
III. Tóm tắt nội dung bài học.....	60
IV. Bài tập.....	60
Bài 7 - Bài thực hành: ĐIỀU KHIỂN LẶP.....	61
I. Bài tập làm theo yêu cầu.....	61
II. Bài tập tự làm.....	63
Bài 8 - MẢNG VÀ CON TRỎ.....	65
I. Mảng.....	65
II. Con trỏ.....	73
IV. Tóm tắt nội dung bài học.....	83
V. Bài tập.....	83
Bài 9 - Bài thực hành: MẢNG VÀ CON TRỎ.....	84
I. Bài tập làm theo yêu cầu.....	84
II. Bài tập tự làm.....	87
Bài 10 - XẤU KÝ TỰ.....	89
I. Khai báo.....	89
II. Nhập xuất chuỗi.....	90
III. Một số hàm xử lý chuỗi.....	91
V. Tóm tắt nội dung bài học.....	95
VI. Bài tập.....	95
Bài 11 - Bài thực hành: XẤU KÝ TỰ.....	96
I. Bài tập làm theo yêu cầu.....	96
II. Bài tập tự làm.....	98

Bài 12 - HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH.....	100
I. Tổ chức chương trình	100
II. Hàm do người dùng định nghĩa	104
III. Con trỏ hàm	122
IV. Đề qui	125
V. Tóm tắt nội dung bài học	129
VI. Bài tập	130
Bài 13 - Bài thực hành: HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH.....	131
I. Bài tập làm theo yêu cầu	131
II. Bài tập tự làm	133
Bài 14 - CẤU TRÚC DỮ LIỆU DO NGƯỜI DÙNG TỰ ĐỊNH NGHĨA.....	135
I. Cấu trúc dữ liệu do người dùng tự định nghĩa	135
II. Ngăn xếp	145
III. Hàng đợi.....	151
IV. Tóm tắt nội dung bài học	154
V. Bài tập	154
Bài 15 - Bài thực hành: CẤU TRÚC DỮ LIỆU DO NGƯỜI DÙNG TỰ ĐỊNH NGHĨA.....	155
I. Bài tập làm theo yêu cầu	155
II. Bài tập tự làm	160
Bài 16 - LÀM VIỆC VỚI FILE.....	162
I. Một số khái niệm	162
II. Các thao tác trên tập tin	163
III. Truy cập tập tin văn bản	165
IV. Truy cập tập tin nhị phân	168
V. Tóm tắt nội dung bài học	172
VI. Bài tập	172
Bài 17 - Bài thực hành LÀM VIỆC VỚI FILE.....	173
I. Bài tập làm theo yêu cầu	173
II. Bài tập tự làm	176
Bài 18 - MỘT SỐ VẤN ĐỀ MỞ RỘNG.....	177
I. Cây, Hàm băm	177
II. Khởi động đồ họa	181
III. Các hàm đồ họa.....	184
IV. Xử lý văn bản trên màn hình đồ họa.....	193
V. Hiệu ứng hoạt hình đồ họa	195
VI. Tóm tắt nội dung bài học	197
VII. Bài tập	197
Bài 19 - Bài thực hành: MỘT SỐ VẤN ĐỀ MỞ RỘNG.....	198
I. Bài tập làm theo yêu cầu	198
II. Bài tập tự làm	198
Bài 20 - ÔN TẬP.....	199
I. Những vấn đề lý thuyết.....	199
II. Các bài thực hành	200

Bài 1 - TỔNG QUAN

Nội dung bài học

- I. Giới thiệu
- II. Bài toán và việc giải bài toán trên máy tính
- III. Công cụ lập trình
- IV. Bài tập

I. Giới thiệu

1. Mục đích, Yêu cầu

Mục đích của môn học là cung cấp cho sinh viên những kiến thức cơ bản về kỹ thuật lập trình nói chung và kỹ năng sử dụng công cụ lập trình C/C++ trong việc giải quyết bài toán bằng chương trình phần mềm trên máy tính.

Kết thúc môn học sinh viên được trang bị những kiến thức về:

- Cách thức giải quyết một bài toán trên máy tính;
- Công cụ, kiểm thức về việc thuật toán hóa bài toán;
- Ngôn ngữ lập trình C/C++;
- Một số cấu trúc dữ liệu và giải thuật điển hình.

Yêu cầu đối với sinh viên

- Có hiểu biết cơ bản về cấu trúc máy tính;
- Có hiểu biết cơ bản về hệ điều hành;
- Biết sử dụng các phần mềm hệ thống trong việc quản lý tài nguyên, sao chép/copy dữ liệu;
- Biết sử dụng các phần mềm ứng dụng như công cụ soạn thảo văn bản, truy cập internet, web.
- Nghe giảng và làm bài tập.

2. Nội dung môn học

Xem chi tiết ở mục lục.

3. Tài liệu học tập và tham khảo

1. Hà Đại Dương, Nguyễn Mậu Uyên, Tập Bài giảng Lập trình cơ bản, HVKTQS 2010;
2. Trần Đức Huyền, Phương pháp giải các bài toán trong tin học, Nhà xuất bản giáo dục 1997;
3. Đào Thanh Tĩnh, Hà Đại Dương, Tin học đại cương, Học viện KTQS, 2003;
4. Đỗ Xuân Lô, Cấu trúc dữ liệu và giải thuật, NXB Giáo dục, 1997;
5. Robert Sedgewick, Algorithms in C++, Addison-Wesley 1992;
6. Niklaus Wirth Bản dịch của Nguyễn Quốc Cường, Cấu trúc dữ liệu + Giải thuật = Chương trình, NXB KHKT, 2001;
7. Giáo trình Tin Học Đại Cương A, Nguyễn Văn Linh, Khoa Công Nghệ Thông Tin, Đại học Cần Thơ, 1991.
8. Giáo trình lý thuyết và bài tập ngôn ngữ C; Nguyễn Đình Tê, Hoàng Đức Hải, Nhà xuất bản Giáo dục, 1999.
9. C - Tham khảo toàn diện, Nguyễn Cẩn, Nhà xuất bản Đồng Nai, 1996.
10. Giúp tự học Lập Trình với ngôn ngữ C, Võ Văn Viện, Nhà xuất bản Đồng Nai, 2002.
11. The C Programming Language, Brian W. Kernighan & Dennis Ritchie, Prentice Hall Publisher, 1988.

II. Bài toán và việc giải bài toán trên máy tính

1. Phương pháp tổng quát để giải một bài toán trên máy tính

Để giải một bài toán trên máy tính cần thực hiện các bước sau:

1. Xác định bài toán;
2. Xác định cấu trúc dữ liệu để mô tả bài toán;
3. Xây dựng thuật toán;
4. Soạn thảo văn bản chương trình, kiểm tra và hoàn thiện chương trình.

2. Xác định bài toán

Khái quát về bài toán

Trong quá trình tồn tại và phát triển, mọi cá nhân luôn phải giải quyết nhiều bài toán đặt ra trong cuộc sống. Có thể nói cuộc sống là một chuỗi các bài toán mà ta phải đối đầu để giải quyết.

Theo nhiều nhà nghiên cứu thì mọi bài toán đều có thể diễn đạt theo một sơ đồ chung như sau:

$$A \Rightarrow B \quad (*)$$

trong đó:

- A là giả thiết, điều kiện ban đầu, thông tin đã cho, đã biết;
- B là kết luận, là mục tiêu cần đạt hoặc cái phải tìm, phải làm ra khi kết thúc bài toán;
- \Rightarrow là suy luận, giải pháp cần xác định hoặc chuỗi các thao tác cần thực hiện để có được kết quả B từ cái đã có A.

Xác định bài toán

Theo sơ đồ trên thì việc xác định bài toán có nghĩa là xác định A, B và nếu có thể thì xác định luôn cả các bước thực hiện để “đi” được từ A đến B.

Bài toán trên máy tính

Tương tự như (*), và

- A gọi là đầu vào (INPUT);
- B gọi là đầu ra (OUTPUT);
- \Rightarrow là CHƯƠNG TRÌNH MÁY TÍNH cho kết quả B với đầu vào A.

Khó khăn

Việc xác định một bài toán trên máy tính thường gặp khó khăn sau:

- Thông tin về A, B thường không rõ ràng và không đầy đủ;
- Thông báo về điều kiện đặt ra cho cách giải (\Rightarrow) thường không được nêu ra một cách minh bạch;

Ví dụ 1: Hãy viết chương trình cho phép giải phương trình bậc 2.

$$A = ???, B = ???$$

Ví dụ 2: Giả sử người A có một số tiền X đem gửi tiết kiệm, lãi suất tháng là L% hỏi rằng sau T tháng thì A có bao nhiêu tiền biết rằng cứ 3 tháng thì tiền lãi được cộng vào gốc.

Ví dụ 3: Bài toán 8 hậu - Hãy tìm cách đặt 8 con hậu trên một bàn cờ vua sao cho không có quân hậu nào có thể ăn quân hậu khác.

Ví dụ 4: Cho dãy số a_1, a_2, \dots, a_n hãy sắp xếp dãy trên theo thứ tự giảm dần.

Ví dụ 5: Hãy xây dựng hệ thống quản lý hồ sơ và kết quả học tập của sinh viên.

...

$$A = ???, B = ???$$

Nhận xét quan trọng

Việc xác định bài toán là rất rất quan trọng, nó ảnh hưởng tới cách thức và chất lượng của việc giải quyết bài toán;

- Một bài toán cho dù được diễn đạt *chi tiết, rõ ràng* vẫn nên giả định là phần lớn thông tin về A, B là tiềm ẩn trong đầu người giải. Thông tin về **A hoặc B thường chỉ là biểu tượng gợi nhớ** đến các thông tin tiềm ẩn.
- Bước đầu tiên để xác định một bài toán là phải phát biểu lại bài toán một cách chính xác theo ngôn ngữ của riêng mình vì đó chính là cách tiếp cận bài toán, hiểu bài toán.
- Bước tiếp là tìm hiểu thông tin Input A, Output B và các mối liên hệ giữa chúng;
- Nên xét một vài trường hợp cụ thể để thông qua đó hiểu được cả bài toán, thấy rõ được các thao tác phải làm. Thực tế cho thấy có những bài toán trong tin học chỉ có thể mô tả được thông qua các ví dụ (như:).

3. Cấu trúc dữ liệu và Giải thuật

Cấu trúc dữ liệu

- Trong khoa học máy tính, **cấu trúc dữ liệu là một cách tổ chức lưu trữ và truy cập dữ liệu** trong máy tính sao cho nó có thể được sử dụng một cách hiệu quả (và phụ thuộc cả vào công cụ lập trình).
- Ví dụ (trong C): Mảng (**Array**), Con trỏ (**Pointer**), Xâu ký tự (**String**), **File, Stack, Queue**
- Thông thường, một cấu trúc dữ liệu được chọn cẩn thận sẽ cho phép thực hiện thuật toán hiệu quả hơn.
- Việc chọn cấu trúc dữ liệu thường bắt đầu từ chọn một cấu trúc dữ liệu trừu tượng. Một cấu trúc dữ liệu được thiết kế tốt cho phép thực hiện nhiều phép toán, sử dụng càng ít tài nguyên, thời gian xử lý và không gian bộ nhớ càng tốt.
- Các cấu trúc dữ liệu được triển khai bằng cách sử dụng các kiểu dữ liệu, các tham chiếu và các phép toán trên đó được cung cấp bởi một ngôn ngữ lập trình.

Thuật toán

- **Thuật toán**, còn gọi là **giải thuật**, là một tập hợp hữu hạn của các chỉ thị hay phương cách được định nghĩa rõ ràng cho việc hoàn tất một số sự việc từ một trạng thái ban đầu cho trước; khi các chỉ thị này được áp dụng triệt để thì sẽ dẫn đến kết quả sau cùng như đã dự đoán.
- Thuật toán là một bộ các qui tắc hay qui trình cụ thể nhằm giải quyết một vấn đề trong một số bước hữu hạn, hoặc nhằm cung cấp một kết quả từ một tập hợp của các dữ kiện đưa vào.

Ví dụ 1: Giả sử có hai bình A và B đựng hai loại chất lỏng khác nhau, A chứa dung dịch Da, B chứa dung dịch Db. Giải thuật để đổi dung dịch Da vào bình B và Db vào A là:

Yêu cầu phải có thêm một bình thứ ba gọi là bình C.

Bước 1: Đổ dung dịch Db vào bình C;

Bước 2: Đổ dung dịch Da vào bình B;

Bước 3: Đổ dung dịch Db vào bình A

Ví dụ 2: Một trong những giải thuật tìm ước chung lớn nhất của hai số a và b là:

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 hai số a và b 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.

Một thuật toán có các tính chất sau:

- **Tính chính xác:** để đảm bảo kết quả tính toán hay các thao tác mà máy tính thực hiện được là chính xác.
- **Tính rõ ràng:** Thuật toán phải được thể hiện bằng các câu lệnh minh bạch; các câu lệnh được sắp xếp theo thứ tự nhất định.

- **Tính khách quan:** Một thuật toán dù được viết bởi nhiều người trên nhiều máy tính vẫn phải cho kết quả như nhau.
- **Tính phổ dụng:** Thuật toán không chỉ áp dụng cho một bài toán nhất định mà có thể áp dụng cho một lớp các bài toán có đầu vào tương tự nhau.
- **Tính kết thúc:** Thuật toán phải gồm một số hữu hạn các bước tính toán.

Trình tự thực hiện các bước của thuật toán

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

1. **Tuần tự (Sequential):** Các công việc được thực hiện một cách tuần tự, công việc này nối tiếp công việc kia.
2. **Cấu trúc lựa chọn (Selection):** Lựa chọn một công việc để thực hiện căn cứ vào một điều kiện nào đó. Có một số dạng như sau:
 - Cấu trúc 1: Nếu <điều kiện> (đúng) thì thực hiện <công việc>
 - Cấu trúc 2: Nếu <điều kiện> (đúng) thì thực hiện <công việc 1>, ngược lại (điều kiện sai) thì thực hiện <công việc 2>
 - Cấu trúc 3: Trường hợp <i> thực hiện <công việc i>
3. **Cấu trúc 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 viết chương trình, người lập trình đã xác định được công việc sẽ lặp bao nhiêu lần.
 - Lặp không xác định: là loại lặp mà khi viết chương trình người lập trình chưa xác định được công việc sẽ lặp bao nhiêu lần. Số lần lặp sẽ được xác định khi chương trình thực thi.

Biểu diễn thuật giải

Ngôn ngữ tự nhiên: Ngôn ngữ tự nhiên là ngôn ngữ của chúng ta đang sử dụng, chúng ta có thể sử dụng ngôn ngữ tự nhiên để mô tả giải thuật giống như các ví dụ ở trên.

Ví dụ: Ta có giải thuật giải phương trình bậc nhất dạng $ax + b = 0$ như sau:

Bước 1: Nhận giá trị của các tham số a, b

Bước 2: Xét giá trị của a xem có bằng 0 hay không?

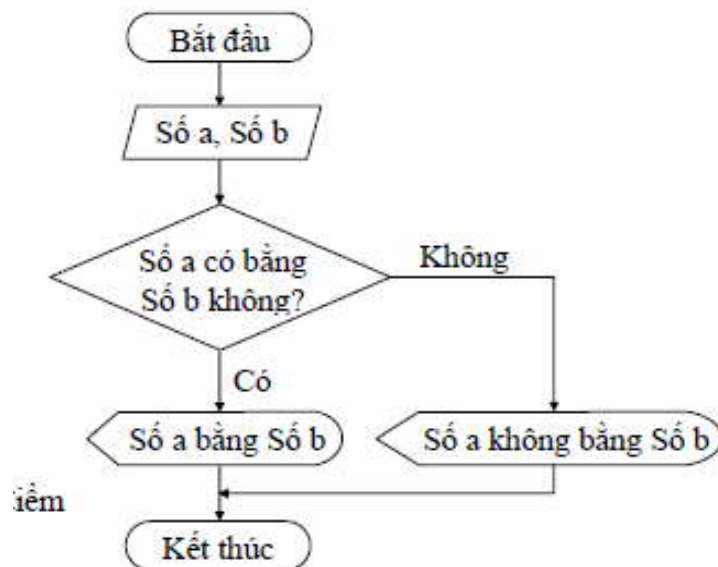
Nếu a=0 thì làm bước 3, nếu a khác không thì làm bước 4.

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) Ta kết luận phương trình có nghiệm $x = -b/a$

Hình dạng (symbol)	Hành động (Activity)
	Dữ liệu vào (Input)
	Xử lý (Process)
	Dữ liệu ra (Output)
	Quyết định (Decision), sử dụng điều kiện
	Luồng xử lý (Flow lines)
	Gọi CT con, hàm... (Procedure, Function...)
	Bắt đầu, kết thúc (Begin, End)
	Điểm ghép nối (Connector)

Lưu đồ thuật toán: Ngôn ngữ sơ đồ (lưu đồ) là một ngôn ngữ đặc biệt dùng để mô tả giải thuật bằng các sơ đồ hình khối. Mỗi khối qui định một hành động như mô tả ở hình trên. Ví dụ: So sánh 2 số.



Giải mã: (tiếng Anh: *Pseudocode*, xuất phát từ chữ pseudo và code) là một bản mô tả giải thuật ngắn gọn và không chính thức, trong đó sử dụng những quy ước có cấu trúc của một số ngôn ngữ lập trình (thường là Pascal) nhưng thường bỏ đi những chi tiết không cần thiết để giúp hiểu rõ giải thuật hơn.

Ví dụ: Thuật giải phương trình bậc 2

Vào: a, b, c

Ra: Kết luận về nghiệm

BEGIN

Delta: = $b*b - 4*a*c$;

If Delta=0 Then

 Phương trình có nghiệm kép $x=-b/(2*a)$;

else

 begin

 if Delta<0 then

 Phương trình Vô nghiệm

 Else

 Begin

 Phương trình có 2 nghiệm

$x1=(-b+\text{sqrt}(\text{Delte})) / (2*a)$

$x2=(-b+\text{sqrt}(\text{Delte})) / (2*a)$

 end

 end

END.

Một số ví dụ khác về thuật toán và biểu diễn thuật toán

Ví 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 $S = a_1 + a_2 + a_3 + \dots + a_n$.

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 $i \leq n$ 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.

Ví 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:

- Bước 1: Nhập 2 số a và b
- Bước 2: Nếu $a = 0$ thì thực hiện bước 3, ngược lại thực hiện bước 4
- Bước 3: Nếu $b=0$ thì thông báo phương trình vô số nghiệm và kết thúc chương trình, ngược lại thông báo phương trình vô nghiệm và kết thúc chương trình.
- Bước 4: Thông báo nghiệm của phương trình là $-b/a$ và kết thúc.

Ví 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, \dots, a_n . Hãy tìm và in ra giá trị lớn nhất trong n số a_1, a_2, \dots, a_n . 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 $i \leq n$ 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

Ví 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, \dots, a_n . Sắp theo thứ tự tăng dần một dãy n số a_1, a_2, \dots, a_n nói trên. Có rất nhiều giải thuật để giải quyết bài toán này. Phần trình bày dưới đây là một phương pháp.

Giả sử ta đã nhập vào máy dãy n số a_1, a_2, \dots, 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 $i \leq n-1$ thì thực hiện bước 4, ngược lại thực hiện bước 8

- Bước 4: Nếu $j \leq n$ 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, \dots, a_n - Kết thúc.

Giải thuật + Cấu trúc dữ liệu = Chương trình

Algorithms + Data Structures = Programs



Niklaus Emil Wirth

In 1975 he wrote the book "Algorithms + Data Structures = Programs", which gained wide recognition and is still useful today.

Biography

Wirth was born in Winterthur, Switzerland, in 1934. In 1959 he earned a degree in Electronics Engineering from the Swiss Federal Institute of Technology Zürich (ETH Zürich). In 1960 he earned an M.Sc. from

Université Laval, Canada. Then in 1963 he was awarded a Ph.D. in EECS from the University of California, Berkeley, supervised by the computer designer pioneer Harry Huskey.

From 1963 to 1967 he served as assistant professor of Computer Science at Stanford University and again at the University of Zurich. Then in 1968 he became Professor of Informatics at ETH Zürich, taking two one-year sabbaticals at Xerox PARC in California (1976–1977 and 1984–1985). Wirth retired in 1999.

Programming languages

Wirth was the chief designer of the programming languages Euler, Algol W, Pascal, Modula, Modula-2, Oberon, Oberon-2, and Oberon-07. He was also a major part of the design and implementation team for the Lilith and Oberon operating systems, and for the Lola digital hardware design and simulation system. He received the ACM Turing Award for the development of these languages and in 1994 he was inducted as a Fellow of the ACM. He designed the simple programming language PL/0 to illustrate compiler design. It has formed the basis for many university compiler design classes.

4. Chương trình

Định nghĩa

- **Chương trình máy tính**: là một tập hợp những câu lệnh được viết bằng một ngôn ngữ lập trình theo một trật tự xác định nhằm tự động thực hiện một số nhiệm vụ hoặc chức năng hoặc giải quyết một bài toán nào đó.
- Là một biểu hiện cụ thể của một giải thuật trên một ngôn ngữ lập trình nào đó cùng với những mô tả về cấu trúc dữ liệu mô tả đầu vào, đầu ra của bài toán cần giải quyết.

Ví dụ 1: Chương trình giải phương trình bậc 2 bằng Pascal

Ví dụ 2: Chương trình giải phương trình bậc 2 bằng C

Tạo ra chương trình máy tính bằng cách nào???

III. Công cụ lập trình

1. Ngôn ngữ lập trình

Khái niệm

- 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. Ví 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 chương trình nguồn (viết bằng ngôn ngữ cấp cao) thành dạng chương trình có khả năng thực thi.

Chương trình dịch

- Như trên đã trình bày, muốn chuyển từ chương trình nguồn sang chương trình đích phải có chương trình dịch. Thông thường mỗi một ngôn ngữ cấp cao đều có một chương trình dịch riêng nhưng chung quy lại thì có hai cách dịch: thông dịch và biên dịch.
- **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 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.

2. Công cụ lập trình

- Trước những năm 1990 người ta cho rằng ngôn ngữ lập trình quyết định kết quả lập trình. Chẳng hạn, trong một tình huống cụ thể nào đó, chương trình viết bằng C++ thì tốt hơn Pascal, viết bằng Pascal thì tốt hơn Fortran... Khi các công cụ lập trình còn thô sơ và các yêu cầu phần mềm chưa cao thì nhận định này là khá chính xác.

- Sau đó người ta cho rằng công nghệ lập trình mới ảnh hưởng lớn nhất đến sản phẩm cuối cùng, sự thống trị trong thập kỷ 90 của lập trình hướng đối tượng và RAD (viết tắt của Rapid Application Development nghĩa là Công cụ phát triển ứng dụng nhanh, thường gọi là lập trình trực quan hay Visual Programming) đã cho thấy tư duy của người lập trình bị ảnh hưởng bởi nền tảng phát triển phần mềm.
- Không ai phê phán Delphi - phiên bản phát triển từ ngôn ngữ Pascal là kém hơn Java hay Visual C++. Tuy mới có 1/20 thời gian của thế kỷ 21 trôi qua nhưng từ đầu thế kỷ đến nay Công nghệ thông tin đã có những bước phát triển mạnh mẽ trong đó có Công nghệ phần mềm. Nhu cầu sử dụng phần mềm và yêu cầu đối với phần mềm đột nhiên tăng vọt khiến nhiều nhà phát triển phần mềm phải xem lại cách làm việc của mình. Đó chính là cơ sở cho sự phát triển rộng rãi trên toàn thế giới của ngành Công nghiệp phần mềm hiện nay.
- Người ta không chỉ quan tâm đến công nghệ lập trình mà còn quan tâm đến quy trình phát triển phần mềm.
- Các công ty bắt đầu chuẩn hóa và đưa ra quy trình công nghệ phần mềm của mình - ở đó việc lựa chọn bộ công cụ lập trình có vai trò rất quan trọng. Các bộ công cụ lập trình hiện nay có xu hướng thống nhất và tương tác với nhau chặt chẽ. Ý tưởng này đã từng xuất hiện trong thập kỷ trước, tiêu biểu như CORBA của Sun hay Delphi - C++ Builder Project Union của Borland, tuy nhiên khi đó chúng chưa được ưa chuộng.
- Khi Visual Studio.NET của Microsoft ra đời năm 2002, người ta nhận thấy rằng các công cụ lập trình nên đi với nhau thành "bộ". Đến thời điểm hiện nay giải pháp về công cụ lập trình của các nhà phát triển luôn được nhắc đến như "bộ công cụ".
- Hiện nay có rất nhiều ngôn ngữ lập trình (NNLT) và rất nhiều công cụ lập trình (CCLT). Một CCLT có thể gắn liền với một NNLT hoặc không - đây là điều mà một số ít người không có kinh nghiệm không hề biết. Ta có thể phân loại:
 - o Theo NNLT: Dòng C có Visual C++, C++ Builder... Dòng Pascal có Borland Pascal, Delphi...
 - o Theo phạm vi sử dụng: Dòng lập trình hệ thống có Microsoft Assembly, Borland C... Dòng lập trình trực quan có Visual Basic, Jbuilder... Dòng lập trình mạng có Java, ASP, PHP...
 - o Theo phong cách lập trình: Dòng cổ điển có Pascal, Fortran... Dòng hướng đối tượng có C++, SmallTalk, Java...
- **Công cụ thường dùng:**
 - o Microsoft Visual Studio 2005, Microsoft Visual Studio 2008: C++, C##, VisualBasic, ASPX...
 - o Java;
 - o PHP;
 - o Pascal, Turbo C, Dev - C++
- **Chức năng của công cụ lập trình:**
 1. Biểu diễn chương trình bằng một ngôn ngữ lập trình nào đó
 2. Giao diện tích hợp (IDE - Itergrated Developments Environment) cho phép soạn thảo văn bản chương trình, kiểm lỗi, thử nghiệm;
 3. Biên dịch thành chương trình độc lập trên máy tính.

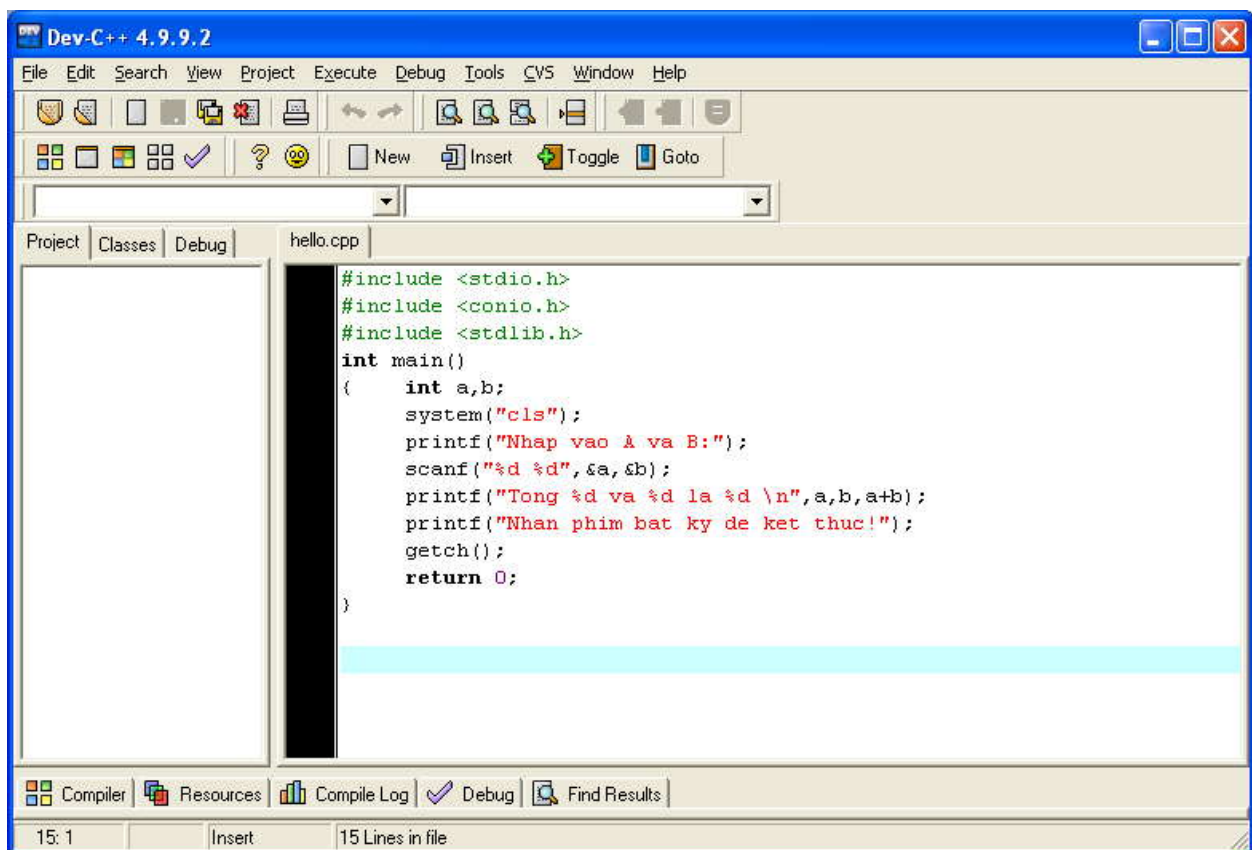
3. Công cụ lập trình Dev-C++

- **Dev-C++ là một công cụ lập trình với giao diện tích hợp cho phép làm việc trên Windows, cho phép lập trình bằng C/C++;**

- Dự án phát triển Dev-C++ được lưu trữ trên SourceForge. Dev-C++ nguyên được phát triển bởi một lập trình viên có tên là Colin Laplace và chỉ chạy trên hệ điều hành Microsoft Windows.
- Bloodshed Dev-C++ là một Môi trường Phát triển Tích hợp (IDE) có hỗ trợ đầy đủ tính năng cho ngôn ngữ lập trình C/C++. Nó sử dụng trình MinGW của GCC (Bộ trình dịch GNU) làm trình biên dịch. Dev-C++ cũng có thể được dùng kết hợp với Cygwin hay bất kỳ trình dịch nền tảng GCC nào khác.
- Chương trình cho người dùng có cảm nhận hơi giống với hình thức của chương trình Microsoft Visual Studio vốn được sử dụng rộng rãi hơn. Dev-C++ có một đặc điểm phụ đó là nó sử dụng DevPaks, là một phần gồm các gói mở rộng so với môi trường tiêu chuẩn, bao gồm các thư viện, mẫu, và các tiện ích được đưa thêm vào. DevPaks thường có, nhưng không nhất định, tiện ích GUI (giao diện người dùng đồ họa), bao gồm các công cụ phổ biến như GTK+, wxWidgets, và FLTK. Có những DevPaks có chứa các thư viện với thậm chí nhiều hàm chức năng cao hơn.
- Dev-C++ nói chung là một chương trình chỉ chạy trên Windows. Tuy nhiên cũng có một phiên bản cho Linux, nhưng vẫn trong giai đoạn alpha và chưa được cập nhật trong vòng hơn 6 năm qua.

Một số giao diện

Giao diện chính



IV. Tóm tắt nội dung bài học

I. Giới thiệu

1. Mục đích, Yêu cầu
2. Nội dung môn học
3. Tài liệu học tập và tham khảo
4. Hình thức đánh giá kết quả môn học

- 5. Tổng quan về máy tính
- II. Bài toán và việc giải bài toán trên máy tính
 - 1. Xác định bài toán
 - 2. Cấu trúc dữ liệu và Giải thuật
 - 3. Chương trình
- III. Công cụ lập trình
 - 1. Ngôn ngữ lập trình
 - 2. Công cụ lập trình
 - 3. Công cụ lập trình Dev-C++

V. Bài tập

- 1. Biểu diễn thuật toán giải các bài toán sau bằng sơ đồ khối và giả mã:
 - a. Giải phương trình bậc 2;
 - b. Giải hệ phương trình bậc nhất;
 - c. Tính số ngày của một tháng một năm nào đó;
 - d. Thuật toán tìm ước số chung lớn nhất;
- 2. Cài đặt, làm quen với giao diện Dev-C++
- 3. Tìm hiểu những bài toán thực tế, mô tả thuật toán để giải quyết vấn đề đó.

Bài 2 - NGÔN NGỮ LẬP TRÌNH C/C++

Nội dung bài học

- I. Giới thiệu
- II. Một số khái niệm cơ bản
- III. Cấu trúc chương trình đơn giản trong C
- IV. Nhập/Xuất dữ liệu
- V. Bài tập

I. Giới thiệu

- **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, Dennis 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, Dennis 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 “hoi 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:**
 - o **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.
 - o **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.
 - o **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.
 - o **Tính linh động (flexible):** C là một ngôn ngữ rất uyển chuyển và cú pháp, chấp nhận nhiều cách thể hiện, có thể thu gọn kích thước của các mã lệnh làm chương trình chạy nhanh hơn.
 - o **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. Một số khái niệm cơ bản

1. Bộ ký tự

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

- 26 chữ cái latin lớn A,B,C...Z ;
- 26 chữ cái latin nhỏ a,b,c ...z. ;
- 10 chữ số thập phân 0,1,2...9. ;
- Các ký hiệu toán học: +, -, *, /, =, <, >, (,)
- Các ký hiệu đặc biệt: ., ;, " ' _ @ # \$! ^ [] { } ... ;
- Dấu cách hay khoảng trống.

2. Từ khóa

Từ khóa là các từ dành riêng (reserved words) của C với mục đích đã được xác định trước. Ví dụ:

- Từ khóa **int** dùng để khai báo biến hoặc hàm với kiểu dữ liệu số nguyên;
- Từ khóa **if** dùng để xây dựng câu lệnh cấu trúc chọn;
- Từ khóa **for** dùng để xây dựng câu lệnh cấu trúc lặp.

Danh sách các từ khóa:

asm	const	else	for	interrupt	return	sizeof	void
break	continue	enum	goto	long	short	switch	volatile
case	default	extern	huge	near	static	typedef	while
cdecl	do	far	if	pascal	struct	union	
char	double	float	int	register	signed	unsigned	

☞ Các từ khóa phải viết bằng *chữ thường*

Lưu ý: không được dùng từ khóa vào mục đích khác, hoặc đặt tên một đối tượng nào đó (biến, hằng, tên hàm ...) trùng với từ khóa. Các từ khóa của Turbo C 3.0 bao gồm:

3. Kiểu dữ liệu

Các kiểu dữ liệu sơ cấp chuẩn trong C có thể được chia làm 2 dạng : kiểu số nguyên, kiểu số thực.

Kiểu số nguyên

Kiểu số nguyên là kiểu dữ liệu dùng để lưu các giá trị nguyên hay còn gọi là kiểu đếm được. Kiểu số nguyên trong C được chia thành các kiểu dữ liệu con, mỗi kiểu có một miền giá trị khác nhau.

Kiểu số nguyên 1 byte (8 bits): Kiểu số nguyên một byte gồm có 2 kiểu sau:

STT	Kiểu dữ liệu	Miền giá trị (Domain)
1	unsigned char	Từ 0 đến 255 (tương đương 256 ký tự trong bảng mã ASCII)
2	char	Từ -128 đến 127

Kiểu unsigned char: lưu các số nguyên dương từ 0 đến 255.

⇒ Để khai báo một biến là kiểu ký tự thì ta khai báo biến kiểu unsigned char. Mỗi số trong miền giá trị của kiểu unsigned char tương ứng với một ký tự trong bảng mã ASCII .

Kiểu char: lưu các số nguyên từ -128 đến 127. Kiểu char sử dụng bit trái nhất để làm bit dấu.

⇒ Nếu gán giá trị > 127 cho biến kiểu char thì giá trị của biến này có thể là số âm (?).

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	enum	Từ -32,768 đến 32,767

2	unsigned int	Từ 0 đến 65,535
3	Short int	Từ -32,768 đến 32,767
4	int	Từ -32,768 đến 32,767

Kiểu enum, short int, int : Lưu các số nguyên từ -32768 đến 32767. Sử dụng bit bên trái nhất để làm bit dấu.

⇒ Nếu gán giá trị >32767 cho biến có 1 trong 3 kiểu trên thì giá trị của biến này có thể là số âm.

Kiểu unsigned int: Kiểu unsigned int lưu các số nguyên dương từ 0 đến 65535.

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	unsigned long	Từ 0 đến 4,294,967,295
2	long	Từ -2,147,483,648 đến 2,147,483,647

Kiểu long : Lưu các số nguyên từ -2147483658 đến 2147483647. Sử dụng bit bên trái nhất để làm bit dấu.

⇒ Nếu gán giá trị >2147483647 cho biến có kiểu long thì giá trị của biến này có thể là số âm.

Kiểu unsigned long: Kiểu unsigned long lưu các số nguyên dương từ 0 đến 4294967295

Kiểu số thực

Kiểu số thực dùng để lưu các số thực hay các số có dấu chấm thập phân gồm có 3 kiểu sau:

STT	Kiểu dữ liệu	Kích thước (Size)	Miền giá trị (Domain)
1	float	4 bytes	Từ $3.4 * 10^{-38}$ đến $3.4 * 10^{38}$
2	double	8 bytes	Từ $1.7 * 10^{-308}$ đến $1.7 * 10^{308}$
3	long double	10 bytes	Từ $3.4 * 10^{-4932}$ đến $1.1 * 10^{4932}$

Mỗi kiểu số thực ở trên đều có miền giá trị và độ chính xác (số số lẻ) khác nhau. Tùy vào nhu cầu sử dụng mà ta có thể khai báo biến thuộc 1 trong 3 kiểu trên.

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

4. Tên, Biến, hằng

Tên

Tên hay còn gọi là danh biểu (identifier) được dùng để đặt cho chương trình, hằng, kiểu, biến, chương trình con... Tên có hai loại là tên chuẩn và tên do người lập trình đặt.

- **Tên chuẩn** là tên do C đặt sẵn như tên kiểu: int, char, float,...; tên hàm: sin, cos...
- **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;
- Độ dài tối đa của tên là không giới hạn, tuy nhiên chỉ có 31 ký tự đầu tiên là có ý nghĩa;
- Không cấm việc đặt tên trùng với tên chuẩn nhưng khi đó ý nghĩa của tên chuẩn không còn giá trị nữa.

Ví dụ: tên do người lập trình đặt:

Chieu_dai, Chieu_Rong, Chu_Vi, Dien_Tich

Tên không hợp lệ: Do Dai, 12A2,...

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 chứa dữ liệu trong quá trình thực hiện chương trình và giá trị của biến có thể bị thay đổi trong quá trình này. Mỗi biến phải thuộc về một kiểu dữ liệu xác định và có miền giá trị thuộc kiểu đó.

Cú pháp khai báo biến:

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

Ví dụ:

```
int a, b, c; /*Ba biến a, b,c có kiểu int*/
long int chu_vi; /*Biến chu_vi 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: Để kết thúc 1 lệnh phải có dấu chấm phẩy (;) ở cuối lệnh.

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 lè). Chúng ta có 2 cách đặt vị trí của biến như sau:

- **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), ví dụ :

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

- **Khai báo biến trong:** Các biến được đặt ở bên trong hàm, chương trình chính hay một khối lệnh. Các biến này chỉ có tác dụng hay ảnh hưởng đến hàm, chương trình hay khối lệnh chứa nó. Khi khai báo biến, phải đặt các **biến này ở đầu của khối lệnh**, trước các lệnh gán, ...

Ví dụ 1:

```
#include <stdio.h>
#include <conio.h>
int bienngoai; /*khai bao bien ngoai*/
int main ()
{
    int j,i; /*khai bao bien ben trong chuong trinh chinh*/
    i=1; j=2;
    bienngoai=3;
    printf("\n Gia tri cua i la %d",i);
        /*%d là số nguyên, sẽ biết sau */
    printf("\n Gia tri cua j la %d",j);
    printf("\n Gia tri cua bienngoai la %d",bienngoai);
    getch();
    return 0;
}
```

Ví dụ 2:

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int i, j; /*Bien ben trong*/
```

```

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 ben trong */
    printf("\n Hieu so cua j tru i la %d",hieu);
}
else
{
    int hieu=i-j ; /*Bien ben trong*/
    printf("\n Gia tri cua i tru j la %d",hieu);
}
getch();
return 0;
}

```

Hằng

Là đại lượng không đổi trong suốt quá trình thực thi của chương trình.

Cú pháp:

```
const <Kiểu dữ liệu> <Tên_hằng> = Giá_trị;
```

Ví dụ:

```
const int heso=10;
```

Hằng số thực

Số thực bao gồm các giá trị kiểu float, double, long double được thể hiện theo 2 cách sau:

- Cách 1: Sử dụng cách viết thông thường mà chúng ta đã sử dụng trong các môn Toán, Lý, ... Điều cần lưu ý là sử dụng dấu thập phân là dấu chấm (.);

Ví dụ: 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.

Ví dụ: 1234.56e-3 = 1.23456 (là số 1234.56 * 10⁻³)
-123.45E4 = -1234500 (là -123.45 * 10⁴)

Hằng số nguyên

Số nguyên gồm các kiểu int (2 bytes) , long (4 bytes) được thể hiện theo những cách sau :

- Hằng số nguyên 2 bytes (int) hệ thập phân: Là kiểu số mà chúng ta sử dụng thông thường, hệ thập phân sử dụng các ký số từ 0 đến 9 để biểu diễn một giá trị nguyên.

Ví dụ: 123, -242

- Hằng số nguyên 2 byte (int) hệ bát phân: Là kiểu số nguyên sử dụng 8 ký số từ 0 đến 7 để biểu diễn một số nguyên.

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

Ví dụ : 0345, -020 (số 345, -20 trong hệ bát phân)

- Hằng số nguyên 2 byte (int) hệ thập lục phân: Là kiểu số nguyên sử dụng 10 ký số từ 0 đến 9 và 6 ký tự A, B, C, D, E, F để biểu diễn một số nguyên.

Cách biểu diễn: 0x<các ký số từ 0 đến 9 và 6 ký tự từ A đến F>

- Hằng số nguyên 4 byte (long): Số long (số nguyên dài) được biểu diễn như số int trong hệ thập phân và kèm theo ký tự l hoặc L. Một số nguyên nằm ngoài miền giá trị của số int (2 bytes) là số long (4 bytes).

Ví dụ: 45345L hay 45345l hay 45345

- Các hằng số còn lại: Viết như cách viết thông thường (không có dấu phân cách giữa 3 số)

Hằng ký tự

Hằng ký tự là một ký tự riêng biệt được viết trong cặp dấu nháy đơn ('). Mỗi một ký tự tương ứng với một giá trị trong bảng mã ASCII. Hằng ký tự cũng được xem như trị số nguyên.

Ví dụ: 'a', 'A', '0', '9'

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)

Hằng chuỗi ký tự

Hằng chuỗi ký tự là một chuỗi hay một xâu ký tự được đặt trong cặp dấu nháy kép ("). Ví dụ: "Ngon ngu lap trinh C", "Khoa CNTT-HVKTQS".

Lưu ý:

- Một chuỗi không có nội dung "" được gọi là chuỗi rỗng;
- Khi lưu trữ trong bộ nhớ, một chuỗi được kết thúc bằng ký tự NULL ('\0': mã Ascii là 0);
- Để biểu diễn ký tự đặc biệt bên trong chuỗi ta phải thêm dấu \ phía trước. Ví dụ: "I'm a student" phải viết "I\'m a student"; "Day la ky tu "dac biet"" phải viết "Day la ky tu \"dac biet\"".

5. 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 hoặc một biểu thức khác.

Trong trường hợp, biểu thức có nhiều toán tử, ta dùng cặp dấu ngoặc đơn () để chỉ định toán tử nào được thực hiện trước.

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

$$(-b + \text{sqrt}(\Delta))/(2*a)$$

Trong đó 2 là hằng; a, b, Delta là biến.

Các toán tử số học

Trong ngôn ngữ C, các toán tử +, -, *, / làm việc tương tự như khi chúng làm việc trong các ngôn ngữ khác. Ta có thể áp dụng chúng cho đa số kiểu dữ liệu có sẵn được cho phép

bởi C. Khi ta áp dụng phép / cho một số nguyên hay một ký tự, bất kỳ phần dư nào cũng bị cắt bỏ. Chẳng hạn, 5/2 bằng 2 trong phép chia nguyên.

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia
%	Chia lấy phần dư
--	Giảm 1 đơn vị
++	Tăng 1 đơn vị

Tăng và giảm (++ & --)

Toán tử ++ thêm 1 vào toán hạng của nó và -- trừ bớt 1. Nói cách khác:

$$x = x + 1 \text{ giống như } ++x$$

$$x = x - 1 \text{ giống như } x--$$

Cả 2 toán tử tăng và giảm đều có thể tiền tố (đặt trước) hay hậu tố (đặt sau) toán hạng. Ví 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. Khi 1 toán tử tăng hay giảm đứng trước toán hạng của nó, C thực hiện việc tăng hay giảm trước khi lấy giá trị dùng trong biểu thức. Nếu toán tử đi sau toán hạng, C lấy giá trị toán hạng trước khi tăng hay giảm nó.

Tóm lại:

$$x = 10$$

$$y = ++x // y = 11$$

Tuy nhiên:

$$x = 10$$

$$y = x++ // y = 10$$

Thứ tự ưu tiên của các toán tử số học: ++ -- sau đó là * / % rồi mới đến + -

Các toán tử quan hệ và các toán tử Logic

Ý tưởng chính của toán tử quan hệ và toán tử Logic là đúng hoặc sai. Trong C mọi giá trị khác 0 được gọi là đúng, còn sai là 0. Các biểu thức sử dụng các toán tử quan hệ và Logic trả về 0 nếu sai và trả về 1 nếu đúng.

Toán tử	Ý nghĩa
Các toán tử quan hệ	
>	Lớn hơn
>=	Lớn hơn hoặc bằng

<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng
!=	Khác
Các toán tử Logic	
&&	AND
	OR
!	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 là Logic

Cao nhất: !
 > >= < <=
 == !=
 &&
 Thấp nhất: ||

Các toán tử Bitwise

Các toán tử Bitwise ý nói đến kiểm tra, gán hay sự thay đổi các Bit thật sự trong 1 Byte của Word, mà trong C chuẩn là các kiểu dữ liệu và biến char, int. Ta không thể sử dụng các toán tử Bitwise với dữ liệu thuộc các kiểu float, double, long double, void hay các kiểu phức tạp khác.

Toán tử	Ý nghĩa
&	AND
	OR
^	XOR
~	NOT

>>	Dịch phải
<<	Dịch trái

Bảng chân trị của toán tử ^ (XOR)

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

Toán tử ? cùng với :

C có một toán tử rất mạnh và thích hợp 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, E3 được ước lượng và trở thành giá trị của biểu thức.

Ví 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

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. Ở đây, chúng ta sẽ đề cập ngắn gọn đến hai toán tử được sử dụng để thao tác với các con trỏ.

Toán tử thứ nhất là &, là một toán tử quy ước trả về địa chỉ bộ nhớ của hệ số của nó.

Ví 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 quy ước trả về giá trị của biến được cấp phát tại địa chỉ theo sau đó.

Ví 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.

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.

Ví 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.

Dấu ngoặc đơn và dấu ngoặc vuông

Trong C, cặp dấu ngoặc đơn là toán tử 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.

Cách viết tắt trong C

Có nhiều phép gán khác nhau, đôi khi ta có thể sử dụng viết tắt trong C nữa. Chẳng hạn:

$x = x + 10$ được viết thành $x += 10$

Toán tử += báo cho chương trình dịch biết để tăng giá trị của x lên 10. Cách viết này làm việc trên tất cả các toán tử nhị phân (phép toán hai ngôi) của C. Tổng quát:

(Biến) = (Biến) (Toán tử) (Biểu thức)

có thể được viết:

(Biến) (Toán tử) = (Biểu thức)

Tổng kết về độ ưu tiên

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

III. Cấu trúc một chương trình đơn giản

1. Cấu trúc chung

Chương trình sau được viết bằng C, cho phép người sử dụng nhập vào 2 số rồi in ra kết quả là tổng 2 số đó.

Xét chương trình sau:

```
/* 1. Khai báo sử dụng thư viện*/
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
/* 2. Khai báo Kiểu dữ liệu*/
// se de cap o bai sau
/* 3. Khai báo hằng */
#define heso 10
/* 4. Khai báo biến */
int a, b;
/* 5. Chương trình chính */
int main()
{ /* Chương trình cho phép nhập vào hai số a và b,
   Tính và in ra tổng hai số đó */
    float ketqua; // Khai báo biến ketqua
    system("cls"); // Xóa màn hình
    printf("Nhập vào A và B:"); // Hiện thị thông báo hướng dẫn nhập liệu
    scanf("%d %d",&a,&b); // Nhập các giá trị cho a và b từ bàn phím
    ketqua =float((a+b)/heso);
    printf("Tổng %d và %d là %0.1f\n",a,b,ketqua); // Hiện thị kết quả ra màn hình
    printf("Nhấn phím bất kỳ để kết thúc!");
    getch(); // Đợi nhận phím bất kỳ để tiếp tục
    return 0;
}
```

Kết quả thực hiện chương trình được như hình sau:



```
C:\Documents and Settings\Duong\My Documents\hel...
Nhập vào A và B:36 71
Tổng 36 và 71 là 107
Nhấn phím bất kỳ để kết thúc!
```

Về cơ bản C không qui định 1 cách chặt chẽ cấu trúc của một chương trình; C chỉ qui định một chương trình phải có hàm **main**, và đồng thời đó là chương trình chính của chương trình. Thông thường một chương trình C gồm các phần:

- Khai báo thư viện;
- Khai báo hằng; hằng;
- Chương trình chính;

2. Khai báo sử dụng thư viện

Phần khai báo sử dụng thư viện:

Cú pháp:

```
#include <tên thư viện> ->
```

Hoặc

```
#include "tên thư viện" ->
```

Ví dụ:

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

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

```
#include <stdlib.h>
```

Xem Help để biết danh sách các include file

3. Khai báo hằng, biến, kiểu

Khai báo (định nghĩa) Hằng

Cú pháp:

```
#define Tên_Hằng Giá_tri
```

Ví dụ:

```
#define heso 10
```

Khai báo (định nghĩa) Biến

Cú pháp:

```
Kiểu_Dữ_liệu Danh_sách_Tên_Biến;
```

Ví dụ:

```
int a, b;
```

4. Một số lệnh đơn

- Lệnh gán
- Lệnh xóa màn hình
- Lệnh dừng chương trình

5. Chú thích

Tất cả các dòng bắt đầu bằng hai dấu chéo (//) được coi là chú thích mà chúng không có bất kỳ một ảnh hưởng nào đến hoạt động của chương trình. Chúng có thể được các lập trình viên dùng để giải thích hay bình phẩm bên trong mã nguồn của chương trình. Trong trường hợp này, dòng chú thích là một giải thích ngắn gọn những gì mà chương trình chúng ta làm.

Trong ngôn ngữ lập trình C, nội dung chú thích phải được viết trong cặp dấu /* và */.

6. Chương trình chính

```
int main ()
```

Dòng này tương ứng với phần bắt đầu khai báo hàm **main**. Hàm **main** là điểm mà tất cả các chương trình C++ bắt đầu thực hiện. Nó không phụ thuộc vào vị trí của hàm này (ở đầu, cuối hay ở giữa của mã nguồn) mà nội dung của nó luôn được thực hiện đầu tiên khi chương trình bắt đầu. Thêm vào đó, do nguyên nhân nói trên, mọi chương trình C++ đều phải tồn tại một hàm **main**.

```
int main()  
{  
.....  
    return 0;  
}
```

Theo sau **main** là một cặp ngoặc đơn bởi vì nó là một hàm. Trong C++, tất cả các hàm mà sau đó là một cặp ngoặc đơn () thì có nghĩa là nó có thể có hoặc không có tham số (không bắt buộc). Nội dung của hàm main tiếp ngay sau phần khai báo chính thức được bao trong các ngoặc nhọn ({ }) như trong ví dụ.

```
return 0;
```

Lệnh **return** kết thúc hàm main và trả về mã đi sau nó, trong trường hợp này là 0. Đây là một kết thúc bình thường của một chương trình không có một lỗi nào trong quá trình thực hiện. Như bạn sẽ thấy trong các ví dụ tiếp theo, đây là một cách phổ biến nhất để kết thúc một chương trình C++.

IV. Nhập/Xuất dữ liệu

1. Nhập dữ liệu từ bàn phím – Hàm scanf()

Là hàm cho phép đọc dữ liệu từ bàn phím và gán cho các biến trong chương trình khi chương trình thực thi. Trong ngôn ngữ C, đó là hàm scanf nằm trong thư viện stdio.h.

Cú pháp:

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

Giải thích:

- Chuỗi định dạng: dùng để qui định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân... Một số định dạng khi nhập kiểu số nguyên, số thực, ký tự.

Định dạng	Ý nghĩa
%[số chữ số]d	Nhập số nguyên có tối đa <số chữ số>
%[số chữ số]f	Nhập số thực có tối đa <số chữ số> tính cả dấu chấm
%c	Nhập một ký tự
Ví 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), 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 chúng ta cần nhập giá trị cho nó. Được viết như sau: **&<tên biến>**.

Vi dụ:

```
scanf("%d",&bien1); /*Doc gia tri cho bien1 co kieu nguyen*/
scanf("%f",&bien2); /*Doc gia tri cho bien2 co kieu thuc*/
scanf("%d%f",&bien1,&bien2);
/*Doc gia tri cho bien1 co kieu nguyen, bien2 co kieu thuc*/
scanf("%d%f%c",&bien1,&bien2,&bien3);
/*bien3 co kieu char*/
```

Lưu ý:

- Chuỗi định dạng phải đặt trong cặp dấu nháy kép (“”).
- Các biến (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 kiểu char được chính xác, nên dùng hàm ***fflush(stdin)*** để loại bỏ các ký tự còn nằm trong vùng đệm bàn phím trước hàm scanf() ;
- Để nhập vào một chuỗi ký tự (không chứa khoảng trắng hay kết thúc bằng khoảng trắng), chúng ta phải khai báo kiểu mảng ký tự hay con trỏ 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().

Một số ví dụ khác:

```
int biennguyen;
float bienthuc;
char bienchar;
char chuoil[20], *chuoil2;
```

1. Lệnh: **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.
2. Lệnh: **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.
3. Lệnh: **scanf("%2d%5f",&biennguyen,&bienthuc);** Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng: 1223 3.142325 thì :
 - a. 2 ký số đầu tiên (12) sẽ được đọc vào cho biennguyen ;
 - b. 2 ký số tiếp theo trước khoảng trắng (23) sẽ được đọc vào cho bienthuc.

4. Lệnh: `scanf("%2d%5f%c", &biennghuyen, &bienthuc, &bienchar);`
 Nếu ta nhập liên tiếp 2 số cách nhau bởi khoảng trắng: 12345 3.142325 thì :
- 2 ký số đầu tiên (12) sẽ được đọc vào cho `biennghuyen` ;
 - 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 `biennghuyen` ;
 - 5 ký số tiếp theo (34567) sẽ được đọc vào cho `bienthuc` ;
 - `bienchar` sẽ có giá trị là ký số tiếp theo '8'.
5. Lệnh: `scanf("%s", chuoi1);` hoặc `scanf("%s", chuoi2);` ; Nếu ta nhập chuỗi như sau: Nguyen Van Huynh ↵ thì giá trị của biến `chuoi1` hay `chuoi2` chỉ là `Nguyen`.
6. Lệnh: `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 Lai ↵ thì giá trị của biến `chuoi1` là Nguyen Van Lai

2. Xuất dữ liệu ra màn hình - Hàm printf()

Hàm `printf` (nằm trong thư viện `stdio.h`) dùng để xuất 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 qui định kiểu dữ liệu, cách biểu diễn, độ rộng, số chữ số thập phân... Một số định dạng khi đối với số nguyên, số thực, ký tự.

Định dạng	Ý nghĩa
<code>%d</code>	Xuất số nguyên
<code>%.số chữ số thập phân] f</code>	Xuất số thực có <số chữ số thập phân> theo quy tắc làm tròn số.
<code>%o</code>	Xuất số nguyên hệ bát phân
<code>%x</code>	Xuất số nguyên hệ thập lục phân
<code>%c</code>	Xuất một ký tự
<code>%s</code>	Xuất chuỗi ký tự
<code>%e</code> hoặc <code>%E</code> hoặc <code>%g</code> hoặc <code>%G</code>	Xuất số nguyên dạng khoa học (nhân 10 mũ x)

Ví 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 xuất giá trị của nó lên màn hình, mỗi biểu thức phân cách nhau bởi dấu phẩy (,).

Ví dụ 1:

```
#include<stdio.h>
#include<conio.h>
int 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);
    getch();
    return 0;
}
```

Kết quả in ra màn hình như sau:



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 được mô tả trong bảng sau:

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	oxHHH		Ký tự có mã ACSII trong hệ thập lục phân là HHH

Ví dụ 2:

```
#include <stdio.h>
#include <conio.h>
int 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();
    return 0;
}
```

3. Ví dụ

Ví dụ 1: Viết chương trình cho phép nhập vào 2 số a và b, trình bày kết quả và phương pháp công 2 số đó theo hình thức sau (với a=876 và b=7655):

876

$$\begin{array}{r}
 + \\
 7655 \\
 \hline
 = 8531
 \end{array}$$

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
// Chuong trinh chinh
int main()
{
    int a,b,tong;
    system("cls");
    printf("Nhap vao a va b:");
    scanf("%d %d",&a,&b);
    printf("Ket qua theo phuong phap cong \n\n");
    tong=a+b;
    printf("%20d\n",a);
    printf("%10s\n","+");
    printf("%20d\n",b);
    printf("%20s\n","-----");
    printf("%20d\n\n",tong);
    printf("Nhan phim bat ky de ket thuc!");
    getch();
    return 0;
}

```

Kết quả thực hiện

```

D:\Giangday\1011ky1\Ky thuat Lap trinh\samples\Lec01_1.exe
Nhap vao a va b:876 7655
Ket qua theo phuong phap cong

          876
+
          7655
-----
          8531

Nhan phim bat ky de ket thuc!_

```

V. Tóm tắt nội dung bài học

- I. Giới thiệu
- II. Một số khái niệm cơ bản
 1. Bộ ký tự
 2. Từ khóa
 3. Kiểu dữ liệu
 4. Tên, Biến, hằng

5. Biểu thức
- III. Cấu trúc chương trình đơn giản trong C
1. Cấu trúc chung
 2. Khai báo sử dụng thư viện
 3. Khai báo hằng, biến, kiểu
 4. Một số lệnh đơn
 5. Chú thích
 6. Chương trình chính
- IV. Nhập/Xuất dữ liệu
1. Nhập dữ liệu từ bàn phím – Hàm scanf()
 2. Xuất dữ liệu ra màn hình – Hàm printf()
 3. ví dụ

VI. Bài tập

Xem **Bài 3 - Bài tập thực hành Môi trường lập trình Dev-C++ và các lệnh vào/ra cơ bản.**

Bài 3 - Bài thực hành: MÔI TRƯỜNG LẬP TRÌNH VÀ CÁC LỆNH VÀO/RA

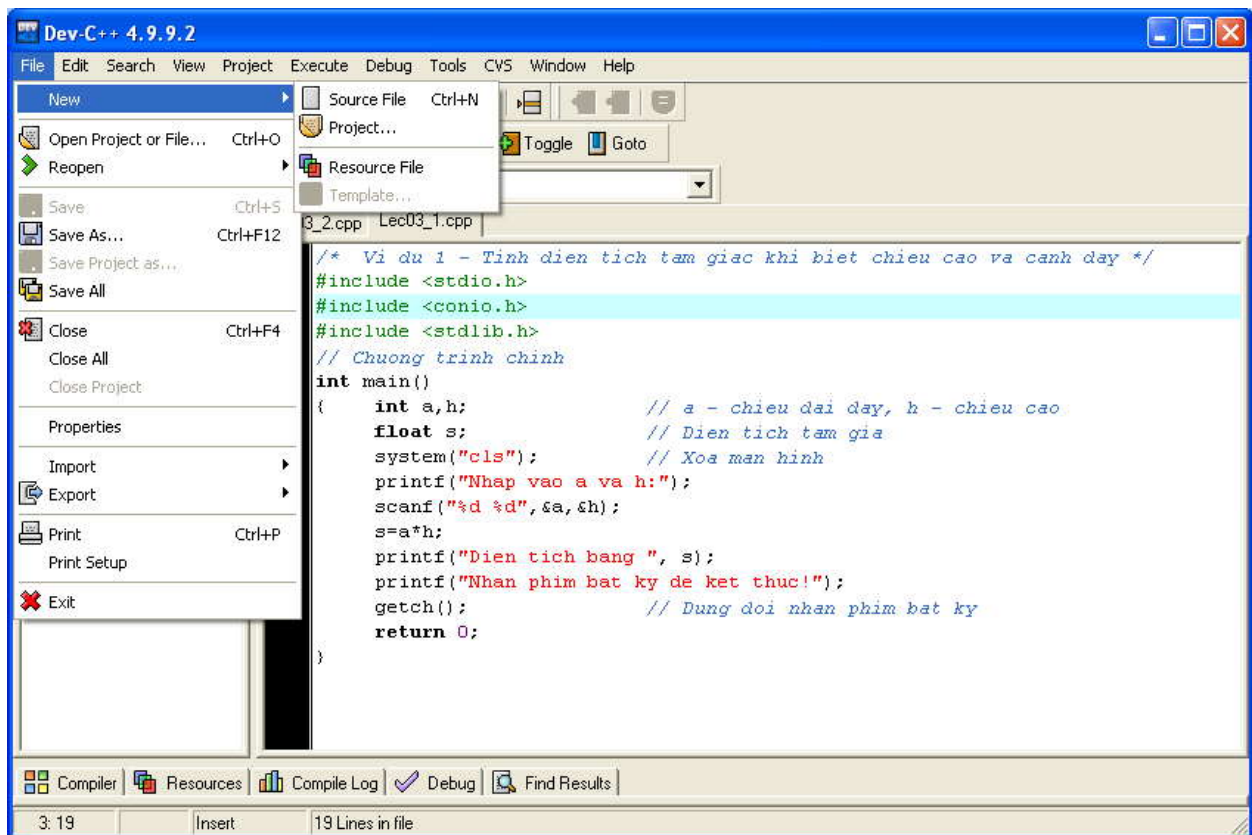
I. Làm quen môi trường Dev-C++

1. Khởi động Dev-C++



Kích đúp vào biểu tượng trên màn hình.

2. Giao diện chính



II. Bài tập làm theo yêu cầu

1. Tính diện tích một tam giác

Yêu cầu: Cho cạnh đáy a một hình tam giác và đường cao tương ứng là h , hãy tính diện tích hình tam giác đó.

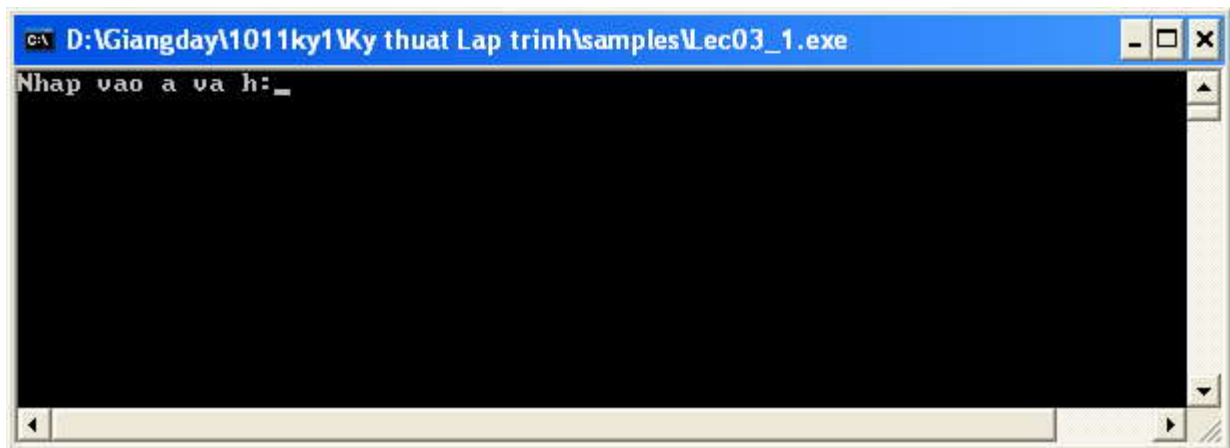
Soạn thảo văn bản chương trình như sau:

```
/* Ví dụ 1 - Tính diện tích tam giác khi biết chiều cao và cạnh đáy */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```
// Chương trình chính
int main()
{   int a,h;           // a - chiều dài đáy, h - chiều cao
    float s;          // Diện tích tam giác
    system("cls");    // Xóa màn hình
    printf("Nhập vào a và h:");
    scanf("%d %d",&a,&h);
    s=a*h;
    printf("Diện tích bằng ", s);
    printf("Nhấn phím bất kỳ để kết thúc!");
    getch();          // Dừng đợi nhận phím bất kỳ
    return 0;
}
```

Thử nghiệm 1:

1. Nhấn **F9**, giao diện xuất hiện như sau:



2. Nhập vào các giá trị: **4** **5 Enter**, khi đó kết quả nhận được như sau:



3. **Nhận xét về kết quả nhận được**

Thử nghiệm 2:

1. Thay dòng `printf("Diện tích bằng ", s);`
 Bằng dòng `printf("Diện tích bằng %f", s);`

2. Nhận xét về kết quả nhận được so với thử nghiệm 1.

Thử nghiệm 3:

1. Thay dòng `printf("Dien tich bang %f", s);`
Bằng dòng `printf("Dien tich bang %10.2f", s);`
2. Nhận xét về kết quả nhận được so với thử nghiệm 2.

2. Tính tổng, hiệu, tích, thương của 2 số

Yêu cầu: Viết chương trình cho phép nhập vào 2 số, tính và in ra màn hình tổng, hiệu, tích và thương của 2 số đó.

Soạn thảo văn bản chương trình như sau

```
/* Chuong trinh tinh TONG, HIEU, TICH, THUONG 2 so*/
```

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

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

```
#include <stdlib.h>
```

```
/* Chuong trinh chinh */
```

```
int main()
```

```
{ /* 1. Khai bao bien va chuan bi*/
```

```
int a,b;
```

```
float tong, hieu,tich, thuong;
```

```
system("cls");
```

```
/* 2. Huong dan nhap va Nhap du lieu vao*/
```

```
printf("Nhap vao A va B:");
```

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

```
/* 3. Tinh toan TONG, HIEU, TICH, THUONG*/
```

```
tong=a+b;
```

```
hieu=a-b;
```

```
tich=a*b;
```

```
thuong=a/b;
```

```
/* 4. Trinh bay ket qua r*/
```

```
printf("Tong %d + %d = %10.2f\n", a,b,tong);
```

```
printf("Hieu %d - %d = %10.2f\n", a,b,hieu);
```

```
printf("Tich %d * %d = %10.2f\n", a,b,tich);
```

```
printf("Thuong %d / %d = %10.2f\n", a,b,thuong);
```

```
printf("Nhan phim bat ky de ket thuc!");
```

```
/* 5. Doi nhan phim bat ky de ket thuc*/
```

```
getch();
```

```
return 0;
```

```
}
```

Thử nghiệm 1:

1. Nhấn **F9**, giao diện xuất hiện như hình sau:



2. Nhập vào các giá trị: **4 5 Enter**, khi đó kết quả nhận được như hình sau.



3. Nhận xét về kết quả nhận được.

Thử nghiệm 2: Cho A=6, B=4 nhận xét về kết quả nhận được;

Thử nghiệm 3: Cho A=6, B=0 nhận xét về kết quả nhận được;

III. Bài tập tự làm

Tự thực hiện các bài tập sau

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

THIỆP MỜI

Than moi ban : Nguyen Van Manh

Toi du le sinh nhat cua minh

Vao luc 19h ngay 12/10/2008

Tai 100 Hoang Quoc Viet – Ha noi

Rat mong duoc don tien !

Ho Thu Huong

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

$$\text{Chu vi CV} = 2 * \text{Pi} * r$$

$$\text{Diện tích S} = \text{Pi} * r * r$$

In các kết quả lên màn hình

4. 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:

$$\text{Chu vi CV} = a + b + c$$

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

$$\text{Trong đó: } p = \text{CV} / 2$$

In các kết quả lên màn hình

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

6. Viết chương trình nhập vào tọa độ của hai điểm (x_1, y_1) và (x_2, y_2)

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

$$\text{Hệ số góc} = (y_2 - y_1) / (x_2 - x_1)$$

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

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

7. 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.

8. Viết chương trình nhập vào các giá trị điện trở R_1 , R_2 , R_3 của một mạch điện :

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

9. 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.

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

11. Viết chương trình đảo ngược một số nguyên dương có đúng 3 chữ số.
12. Cho hai số $x = 100$; $y = 200$; Hãy in ra màn hình các kết quả của các biểu thức:
 $x \& y$; $x \& x$; $!x \& y$; $x \wedge y \wedge y$;
13. Biểu diễn các hằng số nguyên 2 byte sau đây dưới dạng số nhị phân, bát phân, thập lục phân :
- a) 12 b) 255 c) 31000 d) 32767 e) -32768
14. Biểu diễn các hằng ký tự sau đây dưới dạng số nhị phân, bát phân.
- a) 'A' b) 'a' c) 'Z' d) 'z'
15. Cho hệ số k và số x (được biểu diễn ở hệ cơ số k), số k_1 được nhập vào từ bàn phím. Hãy đổi số x từ hệ cơ số k sang hệ cơ số k_1 , biết k, k_1 có thể nhận các số: 2, 8, 10, 16.

Bài 4 - ĐIỀU KHIỂN CHỌN

Nội dung bài học

I. Khối lệnh

II. Lệnh IF

III. Lệnh SWITCH

IV. Ví dụ

V. Bài tập

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.

Ví dụ 1:

```
{
    char ten[30];
    printf("\n Nhập vào ten của bạn:");
    scanf("%s", ten);
    printf("\n Chao Ban %s",ten);
}
```

Ví dụ 2:

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    char ten[50];
    printf("Xin cho biet ten của bạn !");
    scanf("%s",ten);
    getch();
    return 0;
}
```

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:

```
{
    ... lệnh;
    {
        ... lệnh;
        {
            ... lệnh;
        }
        ... lệnh;
    }
    ... lệnh;
}
```

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.

Ví dụ 3:

```
{
    ... 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;
    }
}
```

Ví dụ 4:

```
{
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.

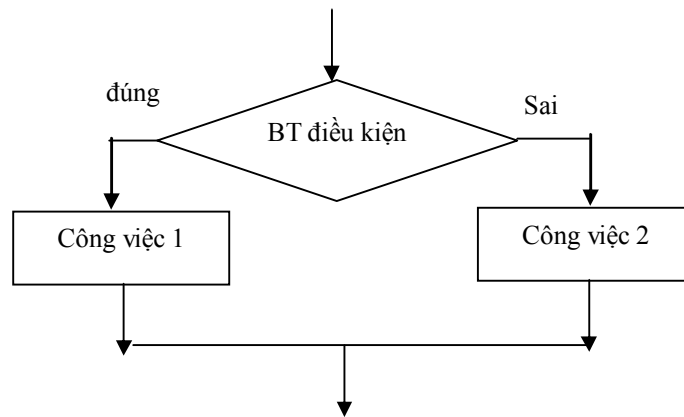
- Một khối lệnh con có thể sử dụng các biến bên ngoài, các lệnh bên ngoài không thể sử dụng các biến bên trong khối lệnh con.

II. Lệnh IF

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.
- Việc có thực hiện công việc 2 hay không là một lựa chọn, có thể có hoặc không.

Ví dụ 1: Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi $a \neq 0$, khi $a = 0$ in ra thông báo “Không thể tìm được nghịch đảo của a”

```

#include <stdio.h>
#include <conio.h>
int 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();
    return 0;
}
  
```

Giải thích:

- Nếu chúng ta 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 `printf("Khong the tim duoc nghich dao cua a")` được thực hiện.
- Lệnh `getch()` luôn luôn được thực hiện.
- Chạy từng bước dạng Debug, view các biến a, b và biểu thức $a > b$

Ví dụ 2: Yêu cầu người chạy chương trình nhập vào giá trị của 2 số a và b, nếu a lớn hơn b thì in ra thông báo “Giá trị của a lớn hơn giá trị của b, giá trị của 2 số”, ngược lại thì in ra màn hình câu thông báo “Giá trị của a nhỏ hơn hoặc bằng giá trị của b, giá trị của 2 số”.

```

#include <stdio.h>
  
```

```

#include<conio.h>
int 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();
    return 0;
}

```

Ví 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

```

#include <stdio.h>
#include<conio.h>
int 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();
    return 0;
}

```

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; Ví dụ:

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

Hoặc:

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

- Nếu sau biểu thức logic của if ta có dấu chấm phẩy, ví dụ:

```
if (a==0) ;
{
printf("Nghịch đảo của %f là %f",a,1/a);
}
```

Thì về cú pháp trình biên dịch sẽ không báo sai, nhưng xét về ý nghĩa là sai.

III. Lệnh SWITCH

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

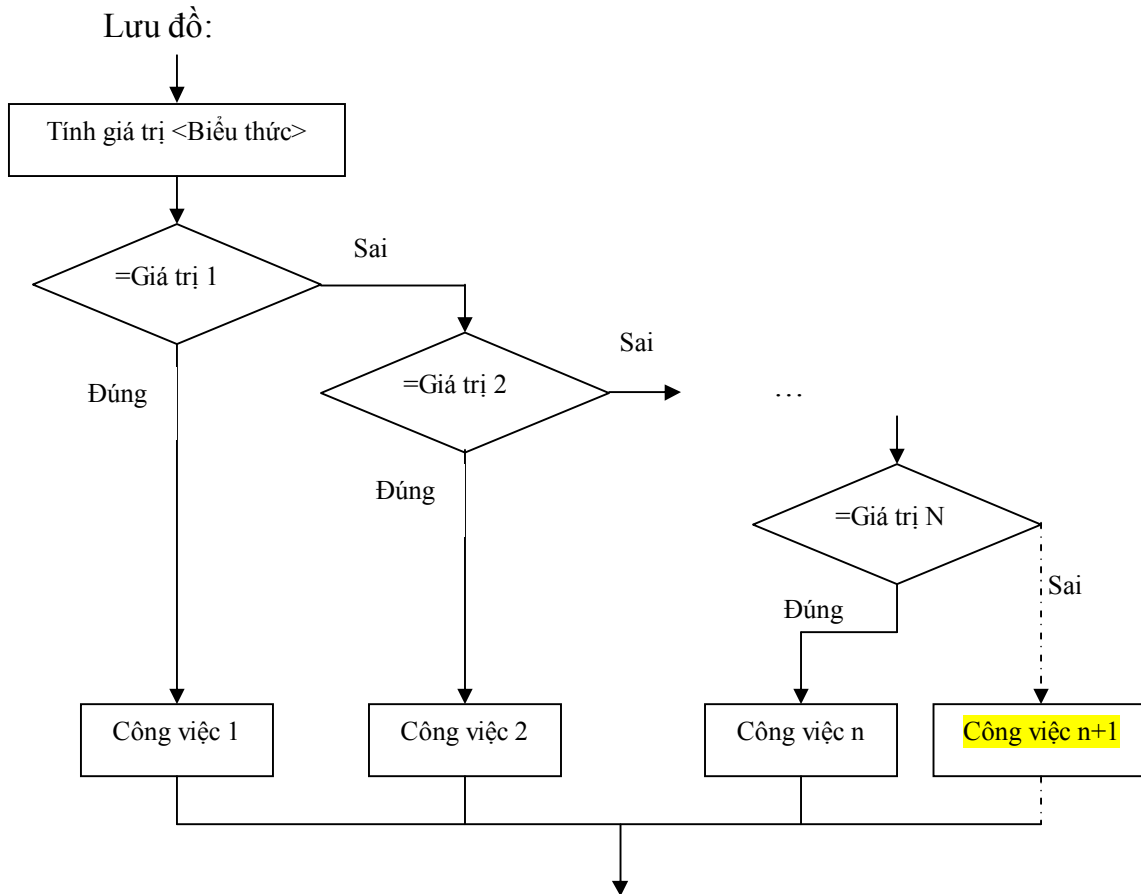
Cú pháp:

```
switch (<Biểu thức>)
{
case giá_trị_1:
{
Công việc 1;
break;
}
...
case giá_trị_n:
{
Công việc N;
break;
}
[default :
{
Công việc N+1;
```

```

        break;
    }
}

```



Giải thích:

- Tính giá trị của **Biểu thức**;
- 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 thì thực hiện **Công việc 2** rồi thoát;
- Tiếp tục so sánh với **Giá trị N** nếu n-1 giá trị trước đó không bằng giá trị của **Biểu thức**.
- Nếu tất cả các phép so sánh trên đều sai thì có thể chọn thực hiện (hay không) **Công việc N+1** (Công việc mặc định).

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;

Ví 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>

```

```

int 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();
    return 0;
}

```

Ví dụ 2: Nhập vào 2 số nguyên và 1 phép toán. Nếu phép toán là '+', '-', '*' thì in ra kết quả 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>
int main ()
{
    int so1, so2;
    float thuong;
    char pheptoan;
    printf("\n Nhap vao 2 so nguyen ");
    scanf("%d%d",&so1,&so2);
    fflush(stdin); //Xoa ky tu enter trong vung dem truoac khi nhap phep toan
    printf("\n Nhap vao phep toan ");
    scanf("%c",&pheptoan);
    switch(pheptoan)
    {
        case '+':
            printf("\n %d + %d =%d",so1, so2, so1+so2);
            break;
        case '-':
            printf("\n %d - %d =%d",so1, so2, so1-so2);
            break;
        case '*':
            printf("\n %d * %d =%d",so1, so2, so1*so2);
            break;
    }
}

```

```

        case '/':
            if (so2!=0){
                thuong=float(so1)/float(so2);
                printf("\n %d / %d =%f", so1, so2, thuong);
            }
            else printf("Khong chia duoc cho 0");
            break;
        default :
            printf("\n Chua ho tro phep toan %c", pheptoan);
            break;
    }
    getch();
    return 0;
}

```

Trong ví dụ trên, tại sao phải xóa ký tự trong vùng đệm trước khi nhập phép toán?

Ví 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>
int main ()
{
    int thang;
    printf("\n Nhap vao thangs 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 Tháng %d có 30 ngày ",thang);
    break;
case 2:
    printf("\ Tháng 2 có 28 hoặc 29 ngày");
    break;
default :
    printf("\n Không có tháng %d", thang);
    break;
}
getch();
return 0;
}
```

IV. Tóm tắt

- I. Khối lệnh
- II. Lệnh IF
- III. Lệnh SWITCH

V. Bài tập

Xem Bài 5 - Bài thực hành Điều khiển chọn

Bài 5 - Bài thực hành: ĐIỀU KHIỂN CHỌN

I. Bài tập làm theo yêu cầu

1. Giải phương trình bậc 2

Yêu cầu: Viết chương trình cho phép biện luận về nghiệm của phương trình bậc 2 $a*x^2+b*x+c = 0$.

Soạn thảo văn bản chương trình như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
int main()
{   int a,b,c;
    float d;
    float x1,x2;
    system("cls");
    printf("Nhap vao cac he so a,b va c:");
    scanf("%d %d %d",&a,&b,&c);
    d=b*b-4*a*c;
    if (d<0)
    {
        printf("Phuong trinh vo nghiem\n");
    }
    else
    {
        if (d==0)
        {
            x1=(-b)/(2*a);
            printf("Phuong trinh co nghiem kep x1=x2=%10.2f\n",x1);
        }
        else
        {
            x1=(-b+sqrt(d))/(2*a);
            x2=(-b-sqrt(d))/(2*a);
            printf("Phuong trinh co 2 nghiem phan biet x1=%10.2f , x2=%10.2f\n",x1,x2);
        }
    }
    printf("Nhan phim bat ky de ket thuc!");
```

```

    getch();
    return 0;
}

```

Thử nghiệm 1:

1. Nhấn **F9**, khi đó giao diện nhập liệu xuất hiện như hình sau:



2. Nhập các giá trị cho a,b, c tương ứng là **2 4 2 Enter**, khi đó kết quả nhận được như hình sau:



3. Nhận xét về kết quả đạt được.

Thử nghiệm 2: Nhập các giá trị cho a,b, c tương ứng là -3 1 2 nhận xét về kết quả nhận được.

Thử nghiệm 3: Nhập các giá trị cho a,b, c bất kỳ, nhận xét về kết quả nhận được.

2. Tạo một menu đơn giản

Soạn thảo văn bản chương trình như sau

```

#include<stdio.h>
#include<conio.h>
int main()
{
    char c;
    while (1==1)

```

```

    {
        c= getch();
        if (c == 'Q') break;
        else printf("%c\n",c);
    }
    return 0;
}

```

Thử nghiệm 1:

- Trước khi chạy chương trình hãy nhận xét xem chương trình thực hiện làm việc gì? Nếu khi chương trình đang chạy người sử dụng nhận phím **Q** (chữ Q hoa) thì chương trình sẽ làm gì?
- Chạy thử chương trình.

Thử nghiệm 2:

- Sửa lại chương trình trên thành như sau:

```

#include<stdio.h>
#include<conio.h>
#include<iostream>
int main()
{
    char c;
    while (1==1)
    {
        system("cls");
        printf("Nhan V de tinh dien tich hinh VUONG\n");
        printf("Nhan C de tinh dien tich hinh CHUNHAT\n");
        printf("Nhan T de tinh dien tich hinh TRON\n");
        printf("-----\n");
        printf("Nhan Q de tinh ket thuc chuong trinh (QUIT)");
        c= getch();
        if (c=='Q') break;
        else
        {
            switch (c)
            {
                case 'V': {
                    // Viet doan tinh dien tich hinh VUONG o day
                }
                break;
                case 'C': {
                    // Viet doan tinh dien tich hinh CHUNHAT o day
                }
                break;
                case 'T': {
                    // Viet doan tinh dien tich hinh TRON o day
                }
                break;
            }
        }
    }
    return 0;
}

```

- Chạy và nhận xét về chương trình

Thử nghiệm 3:

- Giả sử sau dòng ghi chú // **Viết đoạn tính diện tích hình VUONG** o đây thêm vào đoạn chương trình sau:

```
float a, s;  
system("cls");  
printf("Tính diện tích hình VUONG\n");  
printf("-----\n");  
printf("Nhập vào do dai canh: ");  
scanf("%f",&a);  
s= a*a;  
printf("Dein tích hình VUONG canh = %3.1f la: %3.1f\n",a,s);  
printf("Nhập phim bat ky de tro lai menu chính");  
getch();
```

- Chạy và nhận xét về kết quả thực hiện chương trình
- Trong đoạn chương trình trên tại sao phải thêm vào lệnh `system("cls");` ?
- Trong đoạn chương trình trên nếu không có lệnh `getch();` thì vấn đề gì sẽ xảy ra, giải thích.

II. Bài tập tự làm

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. Có hai phương thức gửi tiền tiết kiệm: gửi không kỳ hạn lãi suất 2.4%/tháng, mỗi tháng tính lãi một lần, gửi có kỳ hạn 3 tháng lãi suất 4%/tháng, 3 tháng tính lãi một lần. Viết chương trình tính tổng cộng số tiền cả vốn lẫn lãi sau một thời gian gửi nhập từ bàn phím.

Bài 6 - ĐIỀU KHIỂN LẶP

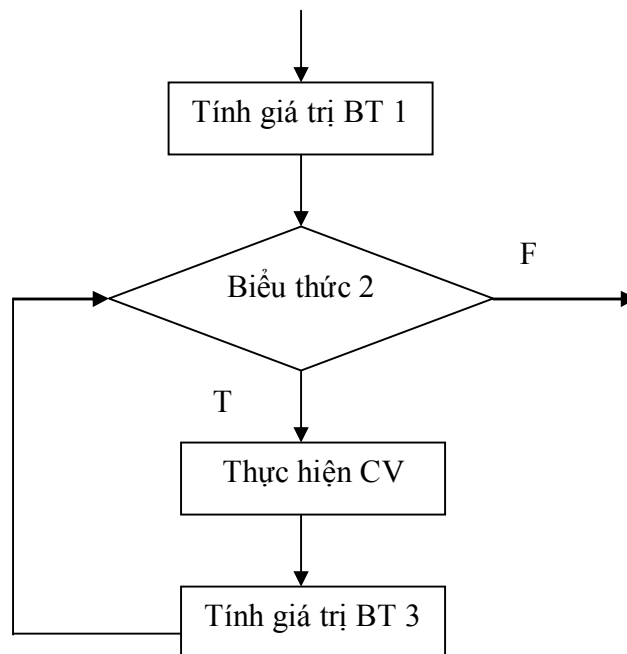
I. Lệnh FOR

Lệnh for cho phép lặp lại công việc cho đến khi điều kiện sai.

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:

B1: Tính giá trị của biểu thức 1.

B2: 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.

B3: Tính giá trị của biểu thức 3 và quay lại B2.

Lưu ý:

- 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 dừng vòng lặp;
- Biểu thức 3: thông thường là một phép gán để thay đổi giá trị của biến điều kiện, biến điều kiện này thường (phải có) trong Biểu thức 2;

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

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

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    int i;
    printf("\n Day so tu 1 den 10 :");
    for (i=1; i<=10; i++)
        {
            printf("%d ",i);
        }
    getch();
    return 0;
}
```

Ví 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>
int main ()
{
    unsigned int n,i,tong;
    printf("\n  Nhập  vào  số  nguyên  dương  n:");
    scanf("%d",&n);
    tong=0;
    for (i=1; i<=n; i++)
        tong+=i;
    printf("\n Tong tu 1 den %d =%d ",n,tong);
    getch();
    return 0;
}
```

Ví 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>
int main ()
{
```

```

unsigned int dong, cot, n, m;
printf("\n Nhap vao so dong va so cot :");
scanf("%d%d", &n, &m);
for (dong=0; dong<n; dong++)
{
    printf("\n");
    for (cot=1; cot<=m; cot++)
        printf("%d\t", dong+cot);
}
getch();
return 0;
}

```

II. Lệnh WHILE

Vòng lặp while giống như vòng lặp for, dùng để lặp lại một công việc nào đó cho đến khi điều kiện sai.

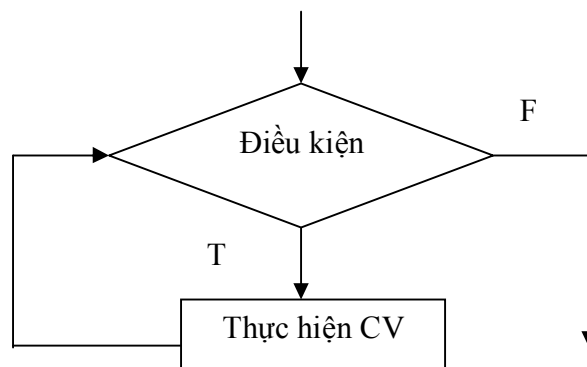
Cú pháp:

```

while (Biểu thức điều kiện)
{
    <Công việc>
}

```

Lưu đồ:



Giải thích:

- <Công việc>: được thể hiện bằng 1 câu lệnh hay 1 khối lệnh;
- Kiểm tra Biểu thức điều kiện;
- 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 nào điều kiện sai;
- Khối lệnh thực hiện công việc có thể rỗng, có thể làm thay đổi điều kiện.

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

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    int i;
    printf("\n Day so tu 1 den 10 :");
    i=1;
    while (i<=10)
        {
            printf("%d ",i);
            i=i+1;
        }
    getch();
    return 0;
}
```

Ví 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>
int main ()
{
    unsigned int n,i,tong;
    printf("\n Nhap vao so nguyen duong n:");
    scanf("%d",&n);
    tong=0;
    i=1;
    while (i<=n)
    {
        tong+=i;
        i++;
    }
    printf("\n Tong tu 1 den %d =%d ",n,tong);
    getch();
    return 0;
}
```

Ví 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>
int main ()
{
    unsigned int dong, cot, n, m;
    printf("\n Nhap vao so dong va so cot :");
    scanf("%d%d",&n,&m);
    dong=0;
    while (dong<n)
    {
        printf("\n");
        cot=1;
        while (cot<=m)
        {
            printf("%d\t",dong+cot);
            cot++;
        }
        dong++;
    }
    getch();
    return 0;
}
```

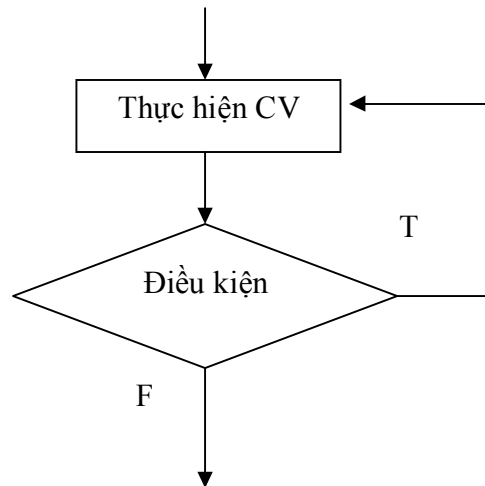
III. Lệnh DO .. WHILE

Vòng lặp do ... while giống như vòng lặp for, while, dùng để lặp lại một công việc nào đó khi 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 Biểu thức điều kiện;
- 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.

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

```

#include <stdio.h>
#include <conio.h>
int main ()
{
    int i;
    printf("\n Day so tu 1 den 10 :");
    i=1;
    do
        printf("%d ",i++);
    while (i<=10);
    getch();
    return 0;
}
  
```

Ví 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>
int main ()
  
```

```

{
    unsigned int n,i,tong;
    printf("\n Nhap vao so nguyen duong n:");
    scanf("%d",&n);
    tong=0;
    i=1;
    do
    {
        tong+=i;
        i++;
    } while (i<=n);
    printf("\n Tong tu 1 den %d =%d ",n,tong);
    getch();
    return 0;
}

```

Vi 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>
int main ()
{
    unsigned int dong, cot, n, m;
    printf("\n Nhap vao so dong va so cot :");
    scanf("%d%d",&n,&m);
    dong=0;
    do
    {
        printf("\n");
        cot=1;
        do
        {
            printf("%d\t",dong+cot);
            cot++;
        } while (cot<=m);
        dong++;
    } while (dong<n);
    getch();
    return 0;
}

```

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 nào 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 nào điều kiện sai.

IV. Lệnh break và continue

Lệnh break

Cú pháp:

break;

Ý nghĩa: 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 --> break sẽ thoát ra khỏi vòng lặp gần nhất. Ngoài ra, break còn được dùng trong cấu trúc lựa chọn switch.

Lệnh continue

Cú pháp:

continue;

Ý nghĩa : 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 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).

III. Tóm tắt nội dung bài học

- I. Lệnh FOR
- II. Lệnh WHILE
- III. Lệnh DO .. WHILE
- IV. Lệnh BREAK và CONTINUE
- V. Bài tập

IV. Bài tập

Xem Bài 7 - Bài tập thực hành Các cấu trúc điều khiển

Bài 7 - Bài thực hành: ĐIỀU KHIỂN LẶP

I. Bài tập làm theo yêu cầu

1. Tìm USCLN của hai số

Yêu cầu: Viết chương trình cho phép nhập vào 2 số nguyên và in ra ước số chung lớn nhất của 2 số đó.

Soạn thảo văn bản chương trình như sau:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
int main()
{   int a,b,t,u,v;
    system("cls");
    printf("Nhap vao cac he so a va b:");
    scanf("%d %d",&a,&b);
    u=a;
    v=b;
    while (a>0)
    {
        if (a<b)
        {
            t=a;      a=b;      b=t;
        }
        a=a-b;
    }
    printf("USCLN cua %d va %d = %d\n", u, v, b);
    printf("Nhan phim bat ky de ket thuc!");
    getch();
    return 0;
}
```

Thử nghiệm 1:

1. Nhấn **F9** để chạy chương trình, khi đó giao diện CT xuất hiện như sau:



2. Nhập các giá trị cho a và b tương ứng là **12 18 Enter**, khi đó kết quả nhận được như sau:



```
D:\Giangday11011ky1Ky thuat Lap trinhWidu1.exe
Nhap vao cac he so a va b:12 18
USCLN cua 12 va 18 = 6
Nhan phim bat ky de ket thuc!
```

3. Nhận xét về kết quả đạt được

Thử nghiệm 2: Viết lại chương trình trên theo thuật toán Euler mô tả như sau:

Input: a, b

Output: d

Mô tả:

- 1) $x = a; y = b;$
- 2) While $y > 0$
 - a) $r = x \bmod y;$
 - b) $x = y;$
 - c) $y = r;$End While
- 3) $d = x;$

2. Tính lãi suất tiền gửi

Vấn đề: Giả sử tiền gửi tiết kiệm được tính với lãi suất là $m\%$ mỗi tháng, sau n tháng thì tiền lãi được cộng vào gốc. Viết chương trình cho phép tính và in ra màn hình số tiền có được sau K tháng gửi tiết kiệm với số tiền gốc ban đầu là T .

Sơ thảo văn bản chương trình sau:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float m,T,lai, Tco;
    int n, K;
    printf("Lai suat (Phan tram) : ");    scanf("%f",&m);
    m=m/100;
    printf("So thang de lai vao goc: ");    scanf("%d",&n);
    printf("So tien gui      : ");        scanf("%f",&T);
    printf("So thang gui      : ");        scanf("%d",&K);
    lai= 0;
    for (int i=1; i<=K; i++)
    {
        lai = lai+ m*T;
        if (i%n==0) ;
    }
```

```

        T=T+lai;
    }
}
Tco = T + lai;
printf("So tien: %0.5f",Tco);
getch();
return 0;
}

```

Thử nghiệm 1:

- Chạy chương trình và cho biết kết quả khi nhập vào m=5, n=3, T=100, K=2. Nhận xét về kết quả nhận được
- Chạy chương trình và cho biết kết quả khi nhập vào m=5, n=3, T=100, K=3. Nhận xét về kết quả nhận được.

Thử nghiệm 2:

- Trong khối lệnh

```

if (i%n==0)
{
    T=T+lai;
}

```

Thêm vào lệnh lai =0; để thành như sau:

```

if (i%n==0)
{
    T=T+lai;
    lai = 0 ;
}

```

- Chạy chương trình và cho biết kết quả khi nhập vào m=5, n=3, T=100, K=3. Nhận xét về kết quả nhận được. Giải thích.
- Chạy chương trình và cho biết kết quả khi nhập vào m=5, n=3, T=100, K=20. Nhận xét về kết quả nhận được. Giải thích.

Thử nghiệm 3:

- Giả sử số tháng gửi là không nguyên, nghĩa là số tháng gửi có thể là 3.5 tháng, 4.3 tháng Nếu thay đổi kiểu biến K là float thay vì int bằng cách thay dòng lệnh

```

int n, K;

```

thành hai dòng lệnh sau:

```

int n ;
float K;

```
- Chạy chương trình và cho biết kết quả khi nhập vào m=5, n=3, T=100, K=3. Nhận xét về kết quả nhận được.
- Debug từng bước và giải thích vì sao chương trình sau khi sửa như trên cho kết quả không đúng.

II. Bài tập tự làm

1. Một số nguyên dương chia hết cho 3 nếu tổng các chữ số của nó chia hết cho 3. Viết chương trình nhập vào một số có 3 chữ số, kiểm tra số đó có chia hết cho 3 dùng tính chất trên.(if)
2. Trò chơi "Oẳn tù tì": trò chơi có 2 người chơi mỗi người sẽ dùng tay để biểu thị một trong 3 công cụ sau: Kéo, Bao và Búa. Nguyên tắc: Kéo thắng bao. Bao thắng búa. Búa thắng kéo. Viết chương trình mô phỏng trò chơi này cho hai người chơi và người chơi với máy. (switch)
3. 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. (if)

4. Viết chương trình nhận vào giờ, phút, giây dạng (hh:mm:ss), từ bàn phím. Cộng thêm một số giây vào và in ra kết quả dưới dạng (hh:mm:ss).
5. Viết chương trình nhập vào ngày tháng năm của một ngày, kiểm tra nó có hợp lệ không.
6. 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. 14. 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"
7. Viết chương trình nhập vào ngày tháng năm của ngày hôm nay, in ra ngày tháng năm của ngày mai.
8. Viết chương trình tính các tổng sau: a) $S=1 + 2 + \dots + n$ b) $S=1/2 + 2/3 + \dots + n/(n+1)$ c) $S= -1 + 2 - 3 + 4 - \dots + (-1)^n n$
9. 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.
10. Viết chương trình đếm số chữ số của một số nguyên n.
11. Tìm số nguyên dương k nhỏ nhất sao cho $2k > n$ với n là một số nguyên dương nhập từ bàn phím.
12. Viết chương trình in ra số đảo ngược của một số nguyên n, với n nhập từ bàn phím.
13. Tính giá trị trung bình của một dãy số thực, kết thúc dãy với -1.
14. 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.
15. Tìm số nguyên dương N nhỏ nhất sao cho $1+1/2+ \dots+1/N > S$, với S nhập từ bàn phím.
16. Viết chương trình tính $P=2*4*6*...*(2n)$, n nhập từ bàn phím.
17. 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) - $[a,b] = a*b/(b,r)$.
18. Viết chương trình nhập vào một số nguyên dương n, in ra màn hình các số nguyên tố $p \leq n$. Số nguyên p gọi là số nguyên tố nếu p chỉ chia hết cho một và chia hết cho bản thân nó.
19. 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: $x_{n+1}=(x_n + a/x_n)/2$ $x_{n+1} - x_n < e$ thì căn bậc hai của a bằng x_{n+1} Nếu: Trong đó e là một hằng số cho trước làm độ chính xác.
20. 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: $(n-1) x_n^n + a - n x_n^{n-1} x_{n+1} = 0$ 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.

Bài 8 - MẢNG VÀ CON TRỎ

Nội dung bài học

I. Mảng

1. Mảng trong C
2. Mảng một chiều
3. Mảng nhiều chiều

II. Con trỏ

1. Khai báo và sử dụng biến con trỏ
2. Con trỏ và mảng
3. Con trỏ và tham số hình thức của hàm

III. Tóm tắt nội dung bài học

I. Mảng

1. 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: ký tự, số, chuỗi ký tự;

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

2. Mảng một chiều

Mảng 1 chiều là một dãy các phần tử có cùng tên gọi, có 1 chỉ số để chỉ thứ tự của phần tử đó trong dãy. Mảng một chiều còn có thể hiểu như một Vector.

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

Cú pháp:

```
<Kiểu> <Tên mảng>[n]
```

Trong đó:

- Tên mảng: đây là một cái tên đặt đúng theo quy tắc đặt tên của danh biểu;
- n: 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 **n** phần tử, phần tử thứ nhất là **tên mảng** **[0]**, phần tử cuối cùng là **tên mảng****[n - 1]**;

Ví 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]

Hình 1: Hình ảnh mảng a trong bộ nhớ

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.

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)**

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.

Ví dụ 1:

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

Trong khai báo này, việc truy xuất các phần tử được chỉ ra trong hình 1. Chẳng hạn phần tử thứ 2 (có vị trí 1) là a[1]...

Ví 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>
int main()
{
    int n,i,j,tam;
    int dayso[]={66,65,69,68,67,70};
    n=sizeof(dayso)/sizeof(int); /*Lay so phan tu*/
```

```

printf("\n Noi dung cua mang ");
for (i=0;i<n;i++)
    printf("%d ",dayso[i]);
getch();
return 0;
}

```

Ví 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. Ta sẽ dùng mảng một chiều để lưu lại các số dư đó.

```

#include<conio.h>
#include<stdio.h>
int main()
{
    unsigned int N;
    unsigned int Du;
    unsigned int NhiPhan[20],K=0;
    int i;
    printf("Nhap vao so nguyen N= ");scanf("%d",&N);
    do
    {
        Du=N % 2;
        NhiPhan[K]=Du;
        K++;
        N = N/2;
    } while(N>0);
    printf("Dang nhi phan la: ");
    for(i=K-1;i>=0;i--)
        printf("%d",NhiPhan[i]);
    getch();
    return 0;
}

```

Ví dụ 4: Nhập vào một dãy n số và sắp xếp các số theo thứ tự tăng. Có rất nhiều giải thuật sắp xếp. 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 rồi tiếp tục so sánh. 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.

```

#include<conio.h>
#include<stdio.h>
int main()

```

```

{
    int b[20], N, i,j,t;
/* 1. Nhập số phần tử của mảng*/
    printf("Số phần tử thực tế của mảng N= ");
    scanf("%d",&N);

/* 2. Nhập giá trị các phần tử của mảng*/
    for(i=0; i< N; i++)
    {
        printf("Phần tử thứ %d: ",i);scanf("%d",&b[i]);
    }

/* 3. Sắp xếp giảm dần*/
    for(i=0;i<N-1;i++)
    {
        for(int j=i+1;j<N;j++)
        {
            if (b[i]>b[j])
            {
                t=b[i];
                b[i]=b[j];
                b[j]=t;
            }
        }
    }

/* 4. In kết quả sau khi sắp xếp*/
    printf("Mảng SAU khi sắp xếp: ");
    for (i=0; i<N;i++)
        printf("%d ",b[i]);

    getch();
    return 0;
}

```

Phiên bản khác của chương trình sử dụng hàm (sẽ học ở bài sau) : viết các hàm Nhập (Nhập các số), SắpXếp (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 Nhập(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;
    for(i=0;i<N-1;i++)
        for(int j=i+1;j<N;j++)
            if (a[i]>a[j])
            {
                t=a[i];
                a[i]=a[j];
                a[j]=t;
            }
}
int main()
{
    int b[20], N;
    printf("So phan tu thuc te cua mang N= ");
    scanf("%d",&N);
    Nhap(b,N);
    printf("Mang vua nhap: ");
    InMang(b,N);
    SapXep(b,N); /* G?i hàm s?p x?p*/
    printf("Mang sau khi sap xep: ");
    InMang(b,N);
    getch();
    return 0;
}

```

3. Mảng nhiều chiều

Mảng nhiều chiều là mảng có từ 2 chiều trở lên. Người ta thường sử dụng mảng nhiều chiều để lưu các ma trận, các tọa độ 2 chiều, 3 chiều...

Khai báo mảng 2 chiều tường minh

Cú pháp:

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

Ví 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[8][9]; /* Khai báo mảng 2 chiều có 8*9 phần tử là số thực*/
```

Trong trường hợp này, ta đã khai báo cho một ma trận có tối đa là 8 dòng, mỗi dòng có tối đa là 9 cột. Hình ảnh của ma trận này được cho trong hình 2:

Dòng\Cột	0	1	2	3	4	5	6	7	8
0	m[0][0]	m[0][1]	m[0][2]	m[0][3]	m[0][4]	M[0][5]	m[0][6]	m[0][7]	m[0][8]
1	m[1][0]	m[1][1]	m[1][2]	m[1][3]	m[1][4]	M[1][5]	m[1][6]	m[1][7]	m[1][8]
2	m[2][0]	m[2][1]	m[2][2]	m[2][3]	m[2][4]	M[2][5]	m[2][6]	m[2][7]	m[2][8]
3	m[3][0]	m[3][1]	m[3][2]	m[3][3]	m[3][4]	m[3][5]	m[3][6]	m[3][7]	m[3][8]
4	m[4][0]	m[4][1]	m[4][2]	m[4][3]	m[4][4]	m[4][5]	m[4][6]	m[4][7]	m[4][8]
5	m[5][0]	m[5][1]	m[5][2]	m[5][3]	m[5][4]	m[5][5]	m[5][6]	m[5][7]	m[5][8]
6	m[6][0]	m[6][1]	m[6][2]	m[6][3]	m[6][4]	m[6][5]	m[6][6]	m[6][7]	m[6][8]
7	m[7][0]	m[7][1]	m[7][2]	m[7][3]	m[7][4]	m[7][5]	m[7][6]	m[7][7]	m[7][8]

Hình 2: Ma trận được mô tả là 1 mảng 2 chiều

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.

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.

Ví 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 ví dụ này, ta sẽ sử dụng hàm để làm ngắn gọn hơn chương trình của ta. Ta sẽ viết các hàm: 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(int 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("%d",&a[i][j]);
            }
}
void InMaTran(int a[][10], int M, int N)
{
    int i,j;
    for(i=0;i<M;i++)
        {
            for(j=0; j< N; j++)
                printf("%d ",a[i][j]);
            printf("\n");
        }
}
/* Cong 2 ma tran A & B ket qua la ma tran C*/
void CongMaTran(int a[][10],int b[][10],int M,int N,int 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];
}
int main()
{
    int a[10][10], b[10][10], M, N;
    int c[10][10];/* Ma tran tong*/
    printf("So dong M= "); scanf("%d",&M);
```

```

printf("So cot N= "); 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();
return 0;
}

```

Vi 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 giả sử 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 $i \in [0 \dots \min(M,N)-1]$.

```

#include<conio.h>
#include<stdio.h>
int main()
{
float a[10][10], T=0;
int M, N, i,j, Min;
printf("Ma tran co bao nhieu dong? ");scanf("%d",&M);
printf("Ma tran co bao nhieu cot? ");scanf("%d",&N);
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]);
}
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");
}
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("Tổng các phần tử ở đường chéo chính là: %f",T);
getch();
return 0;
}

```

II. 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.

Một số hạn chế 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, ngôn ngữ C cung cấp cho ta 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 các biến động không có địa chỉ nhất định nên ta không thể truy cập đến chúng được. Vì thế, ngôn ngữ C lại cung cấp cho ta một loại biến đặc biệt nữa để khắc phục tình trạng này, đó 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;
- Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu, luôn có kích thước cố định là 2 byte.

1. Khai báo và sử dụng biến con trỏ

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**.

Ví 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;
```

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

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

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

```
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.

Các thao tác trên con trỏ

Gán địa chỉ của biến cho biến con trỏ

Toán tử & dùng để định vị con trỏ đến địa chỉ của một biến đang làm việc.

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**.

Ví 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;
```

Lưu ý: Khi gán địa chỉ của biến tĩnh cho 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.

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>
```

Ví dụ 3: Ví dụ sau đây cho phép 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).

Ví dụ 4: Đoạn chương trình sau thấy rõ sự thay đổi này :

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int a,b,*pa,*pb;
    a=2;
    b=3;
    printf("\nGia tri cua bien a=%d \nGia 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 doi gia tri cua *pa*/
    *pb=20; /* Thay doi gia tri cua *pb*/
    printf("\nGia tri moi cua bien a=%d \n Gia tri moi cua bien b=%d ",a,b);
    getch();
    return 0;
}
```

Cấp phát vùng nhớ cho biến con trỏ

Trước khi sử dụng biến con trỏ, ta nên cấp phát vùng nhớ cho biến con trỏ này quản lý địa chỉ. Việc cấp phát được thực hiện nhờ các hàm malloc(), calloc() trong thư viện alloc.h.

Cú pháp:

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.

Ví 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:

0		0	1	2	3	4	5	6	7	8	9
pa (2 byte)		pb (2 byte)									

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

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.

Ví dụ: Trong ví 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*/

```

Giải phóng vùng nhớ cho biến con trỏ : Một vùng nhớ đã cấp phát cho 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.

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

```

free(pa);
free(pb);

```

Một số phép toán trên con trỏ

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

Ví dụ

```
int a, *p, *q ; 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, ví dụ trên được viết lại:

```
int a, *p, *q ; 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.

Ví 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:

0	1	2	3	4	5	6	7	8	9
pa			pc			pb			

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 $pc - pa = 4$.

2. Con trỏ và mảng

Con trỏ và mảng 1 chiều

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. 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.

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>)`

Ví dụ 1: 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 */
voidNhapMang(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 */
voidNhapContro(int a[], int N)
{
    int i;
    for(i=0;i<N;i++)
    {
        printf("Phan tu thu %d: ",i);scanf("%d",a+i);
    }
}
int 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();
return 0;
}

```

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.

Ví dụ 2: Giả sử có khai báo:

```

#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <stdlib.h> /* Them vao so voi phien ban tren DOS*/
int 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();
    return 0;
}

```

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]`

Ví dụ 3: 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 <alloc.h>
#include <stdlib.h>          /* Them vao so voi phien ban tren DOS*/
int 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();
    return 0;
}
```

Con trỏ và mảng nhiều chiều

Ta có thể sử dụng con trỏ thay cho mảng nhiều chiều như sau: 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 bởi contro_int+m;
```

```
a[1][1] được quản lý bởi contro_int+m+1;
```

...

```
a[n-1][m-1] được quản lý bởi contro_int+(n-1)*m + (m-1);
```

Tương tự như thế đối với mảng nhiều hơn 2 chiều.

Ví dụ 4: Sự tương đương giữa mảng 2 chiều và con trỏ.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
int 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 l=");
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 l=\n");
for (i=0;i<20;i++)
    printf("%d ",contro_int[i]);
if (contro_int!=NULL)
    free(contro_int);
getch();
return 0;
}

```

3. Con trỏ và 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.

Ví 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;
}
int main()
{
    int m=20,n=30;
    printf("Truoc khi gọi hàm m= %d, n= %d\n",m,n);
    HoanVi(&m,&n);
    printf("Sau khi gọi hàm m= %d, n= %d",m,n);
    getch();
    return 0;
}

```

}

IV. Tóm tắt nội dung bài học

I. Mảng

1. Mảng trong C
2. Mảng một chiều
3. Mảng nhiều chiều

II. Con trỏ

1. Khai báo và sử dụng biến con trỏ
2. Con trỏ và mảng
3. Con trỏ và tham số hình thức của hàm

V. Bài tập

Xem **Bài 9 - Bài tập thực hành Mảng và Con trỏ**

Bài 9 - Bài thực hành: MẢNG VÀ CON TRỎ

I. Bài tập làm theo yêu cầu

1. Tìm phần tử lớn nhất của mảng

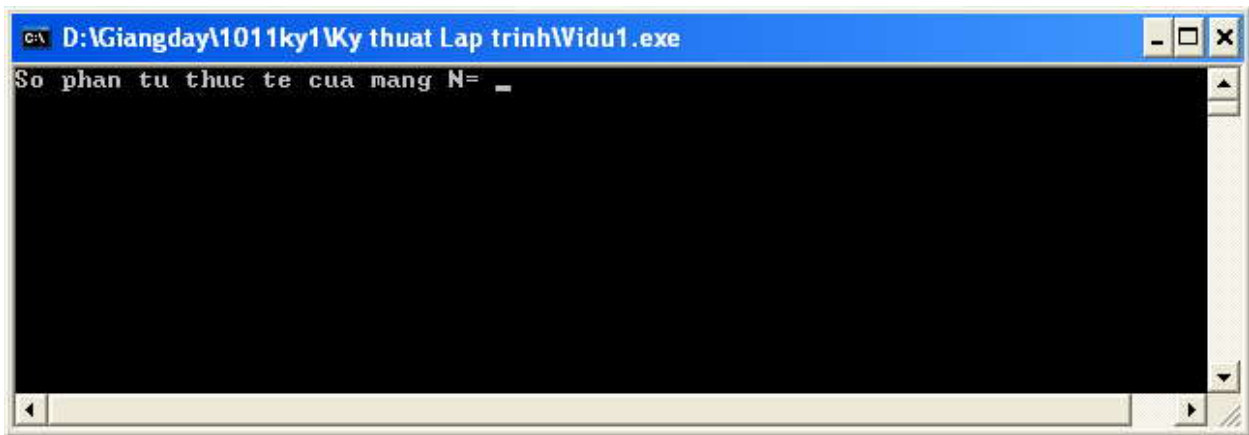
Yêu cầu: Viết chương trình cho phép nhập vào một mảng, tìm phần tử lớn nhất và in ra màn hình

Soạn thảo văn bản chương trình như sau:

```
#include<conio.h>
#include<stdio.h>
int main()
{
    int b[20], N;
    int i, ln;
    printf("So phan tu thuc te cua mang N= ");
    scanf("%d",&N);
    for(i=0; i< N; i++)
    {
        printf("Phan tu thu %d: ",i);scanf("%d",&b[i]);
    }
    ln = b[0];
    for(i=1; i< N; i++)
    {
        if (b[i]>ln)
        {
            ln=b[i];
        }
    }
    printf("Gia tri lon nhat la: %d", ln);
    getch();
    return 0;
}
```

Thử nghiệm 1:

1. Nhấn **F9** để chạy chương trình, khi đó giao diện xuất hiện như sau:



2. Nhập các giá trị cho N (số phần tử của mảng) và các giá trị tương ứng cho mỗi phần tử của mảng như sau: 4 ↵ 8 ↵ 5 ↵ 9 ↵ 1 ↵, khi đó kết quả nhận được như sau:



3. Nhận xét về kết quả đạt được.

Thử nghiệm 2: Đổi việc tìm phần tử lớn nhất thành phần tử nhỏ nhất.

Thử nghiệm 3: Thêm vào yêu cầu là tìm phần tử lớn nhất và vị trí của phần tử đó trong mảng.

2. Tính ma trận tích

Vấn đề: Viết chương trình cho phép nhập vào mảng dữ liệu hai chiều, tính tích hai ma trận và đưa ma trận kết quả ra màn hình.

Soạn thảo văn bản chương trình như sau:

```
#include <stdio.h>
#include <conio.h>
#define N 2
#define M 3
int main()
{
    int i, j, k;
    int a[N][M], b[M][N], c[N][N];
    //nhap du lieu cho mang a
    printf("Nhap ma tran A[%d][%d]:\n", N, M);
    for(i=0; i<N; i++)
```

```

    {
        for(j=0;j<M;j++)
        {
            printf("a[%d][%d]:",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    //nhap du lieu cho mang b
    printf("Nhap ma tran b[%d][%d]:\n",M,N);
    for(i=0;i<M;i++)
    {
        for(j=0;j<N;j++)
        {
            printf("b[%d][%d]:",i,j);
            scanf("%d",&b[i][j]);
        }
    }
    //Nhan hai ma tran
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            c[i][j]=0;
            for(k=0;k<M;k++)
            {
                c[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
    //ghi ra man hinh
    printf("Ma tran tich\n");
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
        {
            printf("%5d",c[i][j]);
        }
        printf("\n");
    }
    getch();
    return 0;
}

```

Thử nghiệm 1:

Nhấn **F9**, nhập hai ma trận sau:

a

1	2	3
3	2	1

b

3	3
2	0
1	-1

- Hãy nhận xét về kết quả nhận được

Thử nghiệm 2:

Thay dòng 49 thành

```
//printf("\n");
```

Nhập lại giá trị ở trên, ghi nhận sự thay đổi kết quả in ra màn hình

Thử nghiệm 3:

Đề nguyên dòng printf("\n");

Thay dòng 47 thành

```
printf("%d",c[i][j]);
```

Nhập lại giá trị ở trên, ghi nhận sự thay đổi kết quả in ra màn hình

Thử nghiệm 4:

- Giữ nguyên các dòng như văn bản ban đầu

- Thay đổi dòng 34 thành

```
//c[i][j]=0;
```

Thực hiện chương trình, nhập dữ liệu mảng ở trên, nhận xét kết quả in ra màn hình

Thử nghiệm 5:

- Giữ nguyên các dòng như văn bản ban đầu

- Thay đổi dòng 4 thành

```
#define M 2
```

Thực hiện chương trình, và nhận xét về số lượng phần tử nhập vào, và kết quả in ra màn hình

II. Bài tập tự làm

- Viết chương trình nhập vào một dãy n số thực $a[0], a[1], \dots, a[n-1]$, sắp xếp dãy số theo thứ tự giảm dần. In dãy số sau khi sắp xếp.
- Viết chương trình sắp xếp một mảng theo thứ tự tăng dần sau khi đã loại bỏ các phần tử trùng nhau.
- 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 .
- 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.
- 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 ?
- 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. Sau đó viết cho ma trận tổng quát cấp $m \times n$.
- 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.
 - Dòng 4 : gồm các số không phải là số nguyên tố.

8. Viết chương trình tính tổng bình phương của các số âm trong một mảng các số nguyên.
9. Viết chương trình thực hiện việc đảo một mảng một chiều.
 Ví dụ : 1 2 3 4 5 7 9 10 đảo thành 10 9 7 5 4 3 2 1 .
 (Không dùng mảng phụ)
10. Viết chương trình nhập vào hai ma trận A và B có cấp m, n. In hai ma trận lên màn hình. Tổng hai ma trận A và B là ma trận C được tính bởi công thức:

$$c_{ij} = a_{ij} + b_{ij} \quad (i=0,1,2,\dots,m-1; j=0,1,2,\dots,n-1)$$
 Tính ma trận tổng C và in kết quả lên màn hình.
11. 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:

$$c_{ij} = a_{i1} * b_{1j} + a_{i2} * b_{2j} + a_{i3} * b_{3j} + \dots + a_{ik} * b_{kj} \quad (i=0,1,2,\dots,m-1; j=0,1,2,\dots,n-1)$$
 Tính ma trận tích C và in kết quả lên màn hình.
12. Xét ma trận A vuông cấp n, các phần tử $a[i, i]$ ($i= 1 \dots n$) 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. Hãy biến đổi ma trận A về ma trận tam giác. In kết quả từng bước lên màn hình.
13. Viết chương trình thực hiện việc trộn hai dãy có thứ tự thành một dãy có thứ tự. Yêu cầu không được trộn chung rồi mới sắp thứ tự. Khi trộn phải tận dụng được tính chất đã sắp của hai dãy con.
14. Thực hiện các bài tập ở phần **Mảng**, bằng cách sử dụng con trỏ.

Bài 10 - XÂU KÝ TỰ

1. Khai báo

- 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 “”.

1. Khai báo theo mảng

Cú pháp:

```
char <Biến> [Chiều dài tối đa];
```

Ví dụ: Trong chương trình, ta có khai báo:

```
char Ten[12];
```

Trong khai báo này, bộ nhớ sẽ cung cấp 12+1 bytes để lưu trữ nội dung của chuỗi ký tự Ten; byte cuối cùng lưu trữ ký tự ‘\0’ để chấm dứt chuỗi.

Lưu ý:

- Chiều dài tối đa của biến chuỗi là một hằng nguyên nằm trong khoảng từ 1 đến 255 bytes.
- Chiều dài tối đa không nên khai báo thừa để tránh lãng phí bộ nhớ, nhưng cũng không nên khai báo thiếu.

2. Khai báo theo con trỏ

Cú pháp:

```
char *<Biến>;
```

Ví 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 “alloc.h”, sau đó mới gán dữ liệu cho biến.

3. Vừa khai báo vừa gán giá trị

Cú pháp:

```
char <Biến>[]=<”Hằng chuỗi”>
```

Ví dụ:

```
#include<stdio.h>
#include<conio.h>
int main()
```

```

{
    char Chuoi[]="Mau nang hay la mau mat em" ;
    printf("Vua khai bao vua gan tri : %s",Chuoi) ;
    getch();
    return 0;
}

```

Lưu ý: 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.

II. Nhập xuất chuỗi

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>);
```

Ví 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.

Ngoài ra, hàm cgets() (trong conio.h) cũng được sử dụng để nhập chuỗi.

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>);
```

Ví dụ: Nhập vào một chuỗi và hiển thị trên màn hình chuỗi vừa nhập.

```

#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Ten[12];
    printf("Nhap chuoi: ");gets(Ten);
    printf("Chuoi vua nhap: ");puts(Ten);
    getch();
    return 0;
}

```

Ngoài ra, ta có thể sử dụng hàm printf(), cputs() (trong conio.h) để hiển thị chuỗi lên màn hình.

III. Một số hàm xử lý chuỗi

Các hàm này có trong string.h

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.

Ví dụ: Nhập vào họ lót và tên của một người, sau đó in cả họ và tên của họ lên màn hình.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int 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);
    getch();
    return 0;
}
```

2. Xác định độ dài chuỗi

- Hàm strlen()

Cú pháp:

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

Ví dụ: Sử dụng hàm strlen xác định độ dài một chuỗi nhập từ bàn phím.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int 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();
    return 0;
}
```

3. Đổi một ký tự thường thành ký tự hoa

- **Hàm toupper()**: Hàm toupper() (trong ctype.h) được dùng để chuyển đổi một ký tự thường thành ký tự hoa.

Cú pháp:

```
char toupper(char c)
```

4. Đổi chuỗi chữ thường thành chuỗi chữ hoa

- **Hàmstrupr()**: Hàmstrupr() 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)
```

Vi dụ: Viết chương trình nhập vào một chuỗi ký tự từ bàn phím. Sau đó sử dụng hàmstrupr() để chuyển đổi chúng thành chuỗi chữ hoa.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Chuoi[255], *s;
    printf("Nhap chuoi: ");
    gets(Chuoi);
    s=strupr(Chuoi) ;
    printf("Chuoi chu hoa: ");
    puts(s);
    getch();
    return 0;
}
```

5. Đổi chuỗi chữ hoa thành chuỗi chữ thường

- **Hàmstrlwr()**: Muốn chuyển đổi chuỗi chữ hoa thành chuỗi toàn chữ thường, ta sử dụng hàmstrlwr(), các tham số của hàm tương tự như hàmstrupr().

Cú pháp:

```
char *strlwr(char *s)
```

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)
```

Ví dụ: Viết chương trình cho phép chép toàn bộ chuỗi nguồn vào chuỗi đích.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Chuoi[255],s[255];
    printf("Nhap chuoi: ");
    gets(Chuoi);
    strcpy(s,Chuoi);
    printf("Chuoi dich: ");
    puts(s);
    getch();
    return 0;
}
```

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)
```

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)
```

Lưu ý :

- 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.

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.

Vi 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>
int main()
{
    char Chuoi[255],*s;
    printf("Nhap chuoi: ");
    gets(Chuoi);
    s=strstr(Chuoi,"hoc");
    printf("Chuoi trích ra: ");
    puts(s);
    getch();
    return 0;
}
```

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).

- 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.

11. So sánh chuỗi

- **Hàm strcmp()**: 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 strcmp(const char *s1, const char *s2)
```

Kết quả trả về tương tự như kết quả trả về của hàm strcmp()

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)
```

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.

V. Tóm tắt nội dung bài học

- I. Giới thiệu
- II. Khai báo
- III. Nhập/Xuất
- IV. Các hàm xử lý

VI. Bài tập

Xem Bài 11 - Bài tập thực hành Xâu ký tự

Bài 11 - Bài thực hành: XÂU KÝ TỰ

I. Bài tập làm theo yêu cầu

1. Tạo chuỗi chuyển động trên màn hình

Vấn đề: Hãy tạo một dòng chữ chuyển động trên màn hình

Soạn thảo đoạn chương trình sau:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX_LENGTH 100
int main()
{
    char st[MAX_LENGTH];
    char sttemp[MAX_LENGTH];
    int length;
    strcpy(st, "Lop hoc lap trinh co ban ");
    length=strlen(st);
    printf("Chuoi truooc khi xu ly (%s)\n",st);
    strcpy(sttemp,&st[1]);
    sttemp[length-1]=st[0];
    sttemp[length]=0;
    strcpy(st,sttemp);
    printf("Chuoi sau khi xu ly (%s)\n",st);
    getch();
    return 0;
}
```

Thử nghiệm 1:

- Dự đoán mục tiêu chương trình
- Chạy thử chương trình.
- Chương trình thực hiện in ra hai dòng dữ liệu khác nhau như thế nào?

Thử nghiệm 2: Soạn thảo đoạn chương trình sau:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#define MAX_LENGTH 100
int main()
{
    int i=0;
    clock_t goal;
    system("cls");
    do
    {
        printf("%d",i);
        i++;
        goal= clock()+100;
        while (goal > clock());
        system("cls");
    } while (!kbhit());
}
```



```
    return 0;
}
```

- Chạy thử chương trình
- Chương trình tiến hành tạo vòng lặp đếm các số và in ra màn hình, mỗi vòng lặp dừng 100 mili giây trước khi in ra số tiếp theo.
- Thử bỏ lệnh system("cls"); xem kết quả

Thử nghiệm 3:

- Phân tích ý nghĩa của

```
goal= clock()+100;
while (goal > clock());
```

trong đoạn chương trình ở trên, thử thay đổi giá trị 100 bởi giá trị khác xem hiệu ứng.

2. Lấy Tên từ họ tên người Việt

Yêu cầu: Giả sử Họ tên người Việt luôn có dạng “Họ Đệm Tên”, trong đó các phần Họ, Đệm, Tên luôn cách nhau 1 dấu cách; phần Đệm có thể có hoặc không hoặc có nhiều hơn 1 từ. Hãy viết chương trình cho phép nhập vào *hoten*, rồi in ra phần *Tên*.

Soạn thảo văn bản chương trình như sau

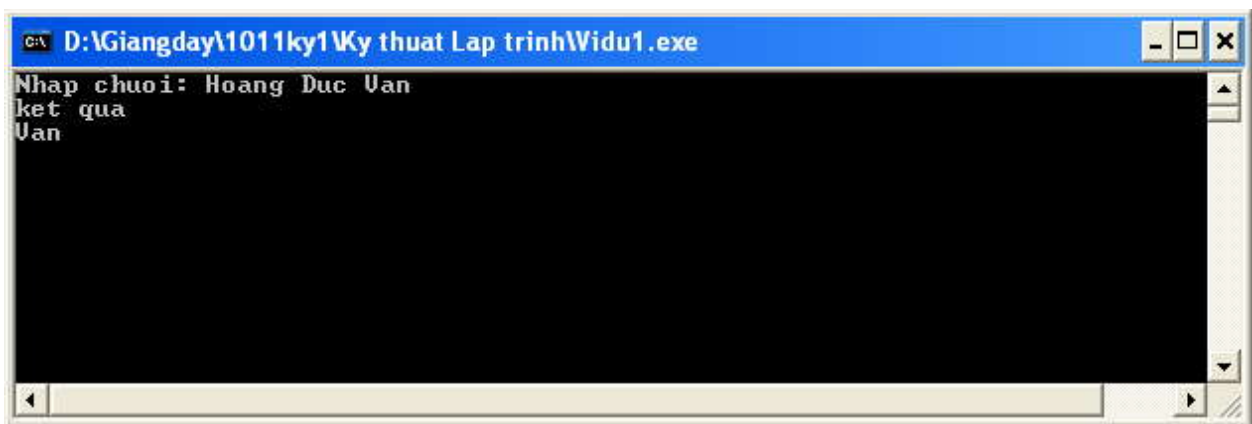
```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Chuoi[255],s[255];
    int n,i,k;
    printf("Nhap chuoi: ");
    gets(Chuoi);
    n=strlen(Chuoi);
    for (i=n-1;i>-1;i--)
    {
        if (Chuoi[i]==32)
        {
            k=i;
            break;
        }
    }
    printf("ket qua\n");
    strncpy(s,Chuoi+k+1,n-k-1);
    puts(s);
    getch();
    return 0;
}
```

Thử nghiệm 1:

1. Nhấn **F9** chạy thử chương trình, khi đó giao diện xuất hiện như hình sau:



2. Nhập giá trị cho chuỗi là **Hoang Duc Van ↵**, khi đó kết quả nhận được như sau:



3. Nhận xét về kết quả đạt được.

Thử nghiệm 2: Chạy và nhập chuỗi vào là “Hoang Duc Van ” (có chứa 2 dấu cách ở cuối), nhận xét về kết quả đạt được.

Thử nghiệm 3: Chạy và nhập chuỗi vào là “HoangDucVan” (không có dấu cách), nhận xét về kết quả đạt được.

II. Bài tập tự làm

- Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình mã Ascii của từng ký tự có trong chuỗi.
- Viết chương trình nhập một chuỗi ký tự từ bàn phím, xuất ra màn hình chuỗi đảo ngược của chuỗi đó. Ví dụ đảo của “abcdx_egh” là “hge_xdcba”.
- Viết chương trình nhập một chuỗi ký tự và kiểm tra xem chuỗi đó có đối xứng không. Ví dụ : Chuỗi ABCDEDCBA là chuỗi đối xứng.
- Nhập vào một chuỗi bất kỳ, hãy đếm số lần xuất hiện của mỗi loại ký tự.
 - Ví dụ:
 - Chuoi1[] = “abcdeaabbdca”
 - Chuoi2[]=”a b c d e”
 - SLXH[] = 4,3,2,2,1

5. Viết chương trình nhập vào một chuỗi.
 - a. In ra màn hình từ bên trái nhất và phần còn lại của chuỗi. Ví dụ: “Nguyễn Văn Minh” in ra thành:
 - i. Nguyễn
 - ii. Văn Minh
 - b. In ra màn hình từ bên phải nhất và phần còn lại của chuỗi. Ví dụ: “Nguyễn Văn Minh” in ra thành:
 - i. Minh
 - ii. Nguyễn Văn
6. Viết chương trình nhập vào một chuỗi rồi xuất chuỗi đó ra màn hình dưới dạng mỗi từ một dòng. Ví dụ: “Nguyễn Văn Minh” in ra :
 - i. Nguyễn
 - ii. Văn
 - iii. Minh
7. Viết chương trình nhập vào một chuỗi, in ra chuỗi đảo ngược của nó theo từng từ. Ví dụ : chuỗi “Nguyễn Văn Minh” đảo thành “Minh Văn Nguyễn”
8. Viết chương trình đổi số tiền từ số thành chữ. Ví dụ: 123 thành chữ là “**mot** tram **hai** muoi ba”.
9. 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.
10. 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.
11. 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.
12. 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.

Bài 12 - HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

Nội dung bài học

I. Tổ chức chương trình

1. Ví dụ
2. Cấu trúc chương trình
3. Hàm xây dựng sẵn

II. Hàm do người dùng định nghĩa

1. Khai báo và định nghĩa Hàm
2. Lời gọi Hàm
3. Hàm với đối mặc định
4. Khai báo hàm trùng tên
5. Truyền tham số
6. Hàm và mảng

III. Con trỏ hàm

1. Khai báo
2. Sử dụng con trỏ hàm
3. Mảng con trỏ hàm

IV. Đệ qui

1. Khái niệm
2. Lớp các bài toán giải được bằng đệ qui
3. Các ví dụ

V. Tóm tắt nội dung bài học

VI. Bài tập

I. Tổ chức chương trình

Mỗi chương trình như đã nêu ra ở các ví dụ trong các chương trước đây thường khá ngắn; do đó:

- Thường không khó để hiểu;
- Dễ nhớ toàn bộ nội dung chương trình cũng như
- Hiểu trình tự logic các bước của công việc.

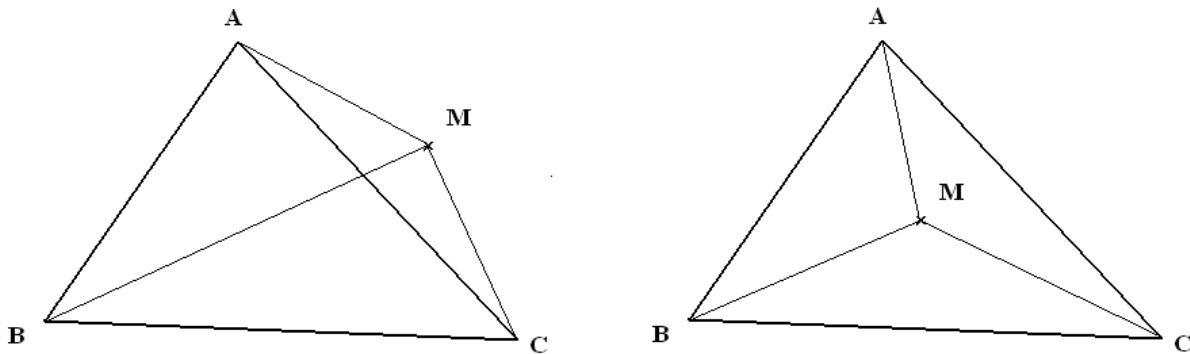
Tuy nhiên khi giải quyết các bài toán thực tế thì văn bản chương trình thường dài hơn rất nhiều, khi đó:

- Việc quản lý trình tự logic các công việc là tương đối khó khăn.
- Thêm nữa, khi viết chương trình chúng ta thường gặp những đoạn chương trình lặp đi lặp lại nhiều lần ở những chỗ khác nhau với sự khác biệt rất nhỏ hoặc thậm chí giống nhau hoàn toàn.

Để giải quyết vấn đề này, tất cả các ngôn ngữ lập trình đều cho phép người sử dụng tổ chức chương trình thành chương trình chính và các chương trình con dạng thủ tục và hàm.

1. Ví dụ

Ví dụ, xét bài toán kiểm tra vị trí tương đối của điểm M trên mặt phẳng so với tam giác ABC là ở trong, nằm trên cạnh hay ngoài tam giác.



Bài toán này có thể giải bằng cách:

- Nếu diện tích tam giác ABC bằng tổng diện tích các tam giác MAB, MBC và MAC thì kết luận là M nằm trong tam giác ABC.
- Ngược lại, khi diện tích tam giác ABC nhỏ hơn tổng diện tích các tam giác MAB, MBC và MAC thì kết luận là M nằm ngoài tam giác ABC.

Nếu theo biện pháp này thì rõ ràng là trong chương trình phải cần ít nhất là bốn lần tính diện tích tam giác. Nếu ta viết một chương trình con tính diện tích tam giác khi biết ba đỉnh U, V, E như DT (U,V,E) chẳng hạn, thì chương trình của chúng ta dường như chỉ còn là một dòng lệnh đơn giản:

```
if (DT (A,B,C) < DT (M,B,C)+DT(M,C,A)+DT(M,A,B))
    printf("M nam ngoai ABC");
else
    printf("M nam trong ABC");
```

Với ví dụ vừa rồi chúng ta thấy rất rõ một lợi ích của việc sử dụng chương trình con là:

- Làm gọn nhẹ chương trình, thay vì phải viết bốn lần cùng một đoạn chương trình rất giống nhau một cách nhàm chán thì giờ đây ta chỉ cần viết có một lần.
- Ngoài ra nó cho phép người lập trình có thể kiểm soát chương trình của mình một cách dễ dàng và thuận tiện hơn.
- Hiển nhiên là việc phải kiểm tra, tìm lỗi logic trong một chương trình có bốn đoạn tính diện tích tam giác so với việc kiểm tra kỹ một đoạn chương trình tính diện tích tam giác cùng với một dòng lệnh rõ ràng và dễ hiểu như trên là rất khác nhau về sự phức tạp.

2. Cấu trúc chương trình

Một chương trình hoàn chỉnh trong C/C++ có 6 phần chính (nhưng không bắt buộc) theo thứ tự như sau:

1. Chỉ thị tiền xử ký;
2. Định nghĩa kiểu dữ liệu;
3. Khái báo prototype;
4. Khai báo biến ngoài;
5. Chương trình chính và
6. Cài đặt hàm.

Nội dung cơ bản các phần này được mô tả chi trong các phần sau đây.

1. Các chỉ thị tiền xử lý

Như đã biết trước khi chạy chương trình (bắt đầu từ văn bản chương trình tức chương trình nguồn) C/C++ sẽ dịch chương trình ra tệp mã máy còn gọi là chương trình đích. Thao tác dịch chương trình nói chung gồm có 2 phần:

- Xử lý sơ bộ chương trình, hay còn gọi là *tiền xử lý* và
- Dịch.

Phần xử lý sơ bộ được gọi là tiền xử lý, trong đó có các công việc liên quan đến các chỉ thị được đặt ở đầu tệp chương trình nguồn như #include, #define ...

Chỉ thị bao hàm tệp #include

Cho phép ghép nội dung các tệp đã có khác vào chương trình trước khi dịch. Các tệp cần ghép thêm vào chương trình thường là các tệp chứa khai báo nguyên mẫu của các hằng, biến, hàm ... có sẵn trong C hoặc các hàm do lập trình viên tự viết. Có hai dạng viết chỉ thị này:

1. #include <tệp>
2. #include "đường dẫn\tệp"

- Dạng khai báo 1 cho phép trình biên dịch tìm tệp cần ghép tại thư mục định sẵn của công cụ lập trình. Thường thì mọi công cụ lập trình dạng C đều xây dựng sẵn các hàm trong các tệp nguyên mẫu, các tệp này được lưu trong thư mục INCLUDES, và thiết lập thư mục mặc định đến thư mục INCLUDES này.
- Dạng khai báo 2 cho phép tìm tệp theo đường dẫn, nếu không có đường dẫn sẽ tìm trong thư mục hiện tại. Tệp thường là các tệp (thư viện) được tạo bởi lập trình viên và được đặt trong cùng thư mục chứa chương trình. Cú pháp này cho phép lập trình viên chia một chương trình thành nhiều môđun đặt trên một số tệp khác nhau để dễ quản lý.

Chỉ thị macro #define

```
#define tên_macro xaukitu
```

- Trước khi dịch bộ tiền xử lý sẽ tìm trong chương trình và thay thế bất kỳ vị trí xuất hiện nào của tên_macro bởi xâu kí tự. Ta thường sử dụng macro để định nghĩa các hằng hoặc thay cụm từ này bằng cụm từ khác để nhớ hơn.

Ví dụ:

```
#define then           // thay then bằng dấu cách
#define begin {       // thay begin bằng dấu {
#define end   }       // thay end bằng dấu }
#define MAX 100        // thay MAX bằng 100
#define TRUE 1         // thay TRUE bằng 1
```

Từ đó trong chương trình ta có thể viết những đoạn lệnh như:

```
if (i < MAX) then
begin
    ok = TRUE;
    printf("%d",i);
end
```

Và trước khi dịch bộ tiền xử lý sẽ chuyển đoạn chương trình trên thành:

```
if (i < 100)
{
    ok = 1;
    printf("%d",i);
}
```

}

theo đúng cú pháp của C/C++ và rồi mới tiến hành dịch.

- Ngoài việc chỉ thị `#define` cho phép thay tên_macro bởi một xâu kí tự bất kỳ, nó còn cũng được phép viết dưới dạng có đối.

Ví dụ, để tìm số lớn nhất của 2 số, thay vì ta phải viết nhiều hàm max (mỗi hàm ứng với một kiểu số khác nhau), bây giờ ta chỉ cần thay chúng bởi một macro có đối đơn giản như sau:

```
#define max(A,B) ((A) > (B) ? (A) : (B))
```

Khi đó trong chương trình nếu có dòng `x = max(a, b)` thì nó sẽ được thay bởi: `x = ((a) > (b) ? (a) : (b))`

- Chú ý:
 - Tên macro phải được viết liền với dấu ngoặc của danh sách đối. Ví dụ không viết `max (A,B)`.
 - `#define bp(x) (x*x)` viết sai vì `bp(5)` đúng nhưng `bp(a+b)` sẽ thành `(a+b*a+b)` (tức `a+b+ab`).
 - Tương tự như trên, viết `#define max(A,B) (A > B ? A: B)` là sai (?) vì vậy luôn luôn bao các đối bởi dấu ngoặc đơn ().
 - `#define bp(x) ((x)*(x))` viết đúng nhưng nếu giả sử lập trình viên muốn tính bình phương của 2 bằng đoạn lệnh sau:

```
int i = 1;  
printf("Ket qua: %d",bp(++i));
```

thì kết quả in ra sẽ là 6 (trên Dev-C++ kết quả này là 9 ?) thay vì kết quả mong muốn là 4. Lí do là ở đây là chương trình dịch sẽ thay `bp(++i)` bởi `((++i)*(++i))`, và với `i = 1` chương trình sẽ thực hiện như `2*3 = 6`. Do vậy cần cẩn thận khi sử dụng các phép toán tự tăng giảm trong các macro có đối. Nói chung, nên hạn chế việc sử dụng các macro phức tạp, vì nó có thể gây nên những hiệu ứng phụ khó kiểm soát.

Các chỉ thị biên dịch có điều kiện `#if`, `#ifdef`, `#ifndef`

1. `#if` dãy lệnh ... `#endif`
2. `#if` dãy lệnh ... `#else` dãy lệnh ... `#endif`,
3. `#ifdef` và `#ifndef`

2. Định nghĩa kiểu dữ liệu

Dùng để đặt tên lại cho một kiểu dữ liệu nào đó để gọi nhớ hay đặt 1 kiểu dữ liệu cho riêng mình dựa trên các kiểu dữ liệu đã có; phần này không bắt buộc phải có. Trong C/C++ cho phép định nghĩa kiểu dữ liệu với từ khóa `struc` và `typedef`; nội dung chi tiết về vấn đề này được trình bày trong phần chương 6.

3. Khai báo các prototype

Khai báo mô tả các hàm sẽ dùng trong chương trình chính bao gồm tên hàm, các tham số hình thức, kiểu dữ liệu trả về của hàm. Phần này là không bắt buộc, tuy nhiên những hàm được dùng trong chương trình chính hoặc chương trình con khác bắt buộc phải được khai báo prototype trước khi sử dụng. Tại vị trí này có thể khai báo đầy đủ một hàm bao gồm cả phần mô tả và thân

hàm, tuy nhiên cách viết đó làm cho chương trình chính bị đẩy sâu xuống cuối chương trình làm cho chương trình khó theo dõi. Vì vậy việc viết các hàm, chương trình con thường được tổ chức thành 2 phần là phần khai báo prototype trước hàm main và phần cài đặt nội dung hàm sau hàm main. Nội dung chi tiết về công việc này sẽ được đề cập cụ thể trong phần sau của chương này.

4. Khai báo các biến ngoài (các biến toàn cục)

Cho phép khai báo các biến, hằng có phạm vi tác động trong toàn bộ chương trình. Phần này là tùy vào nhu cầu xử dụng trong mỗi chương trình mà có thể có hoặc không.

5. Chương trình chính

Phần này bắt buộc phải có. Trong C/C++ chương trình chính được qui định có tên là main. Phần này đã được mô tả trong các phần sau.

6. Cài đặt các hàm

Đây là phần mã nguồn đầy đủ của các hàm. Nội dung này sẽ được nghiên cứu trong các phần tiếp theo của chương này.

3. Hàm xây dựng sẵn

7. **stdio.h**: Thư viện chứa các hàm vào/ ra chuẩn (standard input/output). Gồm các hàm printf(), scanf(), getc(), putc(), gets(), puts(), fflush(), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()...
8. **conio.h**: Thư viện chứa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),...
9. **math.h**: Thư viện chứa các hàm tính toán gồm các hàm abs(), sqrt(), log(), log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),...
10. **alloc.h**: Thư viện chứa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), ...
11. **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(),...
12. **graphics.h**: Thư viện chứa các hàm liên quan đến đồ họa. Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), ...

II. Hàm do người dùng định nghĩa

Hàm nhận (hoặc không) các đối số và trả lại (hoặc không) một giá trị cho chương trình gọi nó. Trong trường hợp không trả lại giá trị, hàm hoạt động như một thủ tục trong các ngôn ngữ lập trình khác. Một chương trình là tập các hàm, trong đó có một hàm chính với tên gọi main(), khi chạy chương trình, hàm main() sẽ được chạy đầu tiên và gọi đến hàm khác. Kết thúc hàm main() cũng là kết thúc chương trình.

Hàm giúp cho việc phân đoạn chương trình thành những môđun riêng rẽ, hoạt động độc lập với ngữ nghĩa của chương trình lớn, có nghĩa một hàm có thể được sử dụng trong chương trình này mà cũng có thể được sử dụng trong chương trình khác, để cho việc kiểm tra và bảo trì chương trình. Hàm có một số đặc trưng:

- Nằm trong hoặc ngoài văn bản có chương trình gọi đến hàm. Trong một văn bản có thể chứa nhiều hàm;

- Được gọi từ chương trình chính (main), từ hàm khác hoặc từ chính nó (đệ quy);
- Không lồng nhau;
- Có 3 cách truyền giá trị: Truyền theo tham trị, tham biến và tham trỏ.

1. Khai báo và định nghĩa Hàm

1. Khai báo

Một hàm thường làm chức năng:

- Tính toán trên các tham đối và cho lại giá trị kết quả, hoặc;
- Chỉ đơn thuần thực hiện một chức năng nào đó, không trả lại kết quả tính toán.

Thông thường kiểu của giá trị trả lại được gọi là kiểu của hàm. Các hàm thường được khai báo ở đầu chương trình. Các hàm viết sẵn được khai báo trong các file nguyên mẫu *.h. Do đó, để sử dụng được các hàm này, cần có chỉ thị #include <*.h> ở ngay đầu chương trình, trong đó *.h là tên file cụ thể có chứa khai báo của các hàm được sử dụng (ví dụ để sử dụng các hàm toán học ta cần khai báo file nguyên mẫu math.h).

Khai báo một hàm như sau:

<kiểu giá trị trả lại> <tên hàm>(d/s kiểu đối) ;

- Trong đó, kiểu giá trị trả lại còn gọi là kiểu hàm và có thể nhận kiểu bất kỳ chuẩn của C++ và cả kiểu của NSD tự tạo. Đặc biệt nếu hàm không trả lại giá trị thì kiểu của giá trị trả lại được khai báo là void. Nếu kiểu giá trị trả lại được bỏ qua thì chương trình ngầm định hàm có kiểu là int (phân biệt với void!).
- Ví dụ:

```
int bp(int);           // Khai báo hàm bp, có đối kiểu int và kiểu hàm là int
int rand100();        // Không đối, kiểu hàm (giá trị trả lại) là int
void alltrim(char[]); // đối là chuỗi ký tự, hàm không trả lại giá trị (không kiểu).
cong(int, int);       // Hai đối kiểu int, kiểu hàm là int (ngầm định).
```

Thông thường để chương trình được rõ ràng chúng ta nên tránh lạm dụng các ngầm định. Ví dụ trong khai báo cong(int, int); nên khai báo rõ cả kiểu hàm (trong trường hợp này kiểu hàm ngầm định là int) như sau : int cong(int, int);.

2. Định nghĩa hàm

Cấu trúc một hàm bất kỳ được bố trí cũng giống như hàm main() trong các phần trước.

Hàm có trả về giá trị

```
<kiểu hàm> <tên hàm>(danh sách tham đối hình thức)
{
    khai báo cục bộ của hàm ; // chỉ dùng riêng cho hàm này
    dãy lệnh của hàm ;
    return (biểu thức trả về); // có thể nằm đâu đó trong dãy lệnh.
}
```

- Trong đó:
 - Danh sách tham đối hình thức còn được gọi ngắn gọn là danh sách đối gồm dãy

các đối cách nhau bởi dấu phẩy, đối có thể là một biến thường, biến tham chiếu hoặc biến con trỏ, hai loại biến sau ta sẽ trình bày trong các phần tới. Mỗi đối được khai báo giống như khai báo biến, tức là cặp gồm <kiểu đối> <tên đối>.

- Với hàm có trả lại giá trị cần có câu lệnh return kèm theo sau là một biểu thức. Kiểu của giá trị biểu thức này chính là kiểu của hàm đã được khai báo ở phần tên hàm. Câu lệnh return có thể nằm ở vị trí bất kỳ trong phần câu lệnh, tùy thuộc mục đích của hàm. Khi gặp câu lệnh return chương trình tức khắc thoát khỏi hàm và trả lại giá trị của biểu thức sau return như giá trị của hàm.
- Ví dụ : Ví dụ sau định nghĩa hàm tính lũy thừa n (với n nguyên) của một số thực bất kỳ. Hàm này có hai đầu vào (đối thực x và số mũ nguyên n) và đầu ra (giá trị trả lại) kiểu thực với độ chính xác gấp đôi là x^n .

```
double luythua(float x, int n)
{
    int i;                // biến chỉ số
    double kq = 1;       // để lưu kết quả
    for (i=1; i<=n; i++) kq *= x;
    return kq;
}
```

Hàm không trả về giá trị

```
void <tên hàm>(danh sách tham đối hình thức)
{
    khai báo cục bộ của hàm ; // chỉ dùng riêng cho hàm này
    dãy lệnh của hàm ;
    return;                // có thể nằm đâu đó trong dãy lệnh.
}
```

- Nếu hàm không trả lại giá trị (tức kiểu hàm là void), khi đó có thể có hoặc không có câu lệnh return, nếu có thì đằng sau return sẽ không có biểu thức giá trị trả lại.
- Ví dụ: Hàm xoá màn hình 100 lần, hàm chỉ làm công việc cần thận xoá màn hình nhiều lần để màn hình thật sạch, nên không có giá trị gì để trả lại.

```
void xmh()
{
    int i;
    for (i=1; i<=100; i++)
        clrscr();
    return ;
}
```

- Hàm main() thông thường có hoặc không có giá trị trả về cho hệ điều hành khi chương trình chạy xong, vì vậy ta thường khai báo kiểu hàm là int main() hoặc void main() và câu lệnh cuối cùng trong hàm thường là return 1 hoặc return. Trường hợp bỏ qua từ khoá void nhưng trong thân hàm không có câu lệnh return (giống phần lớn ví dụ trong giáo trình này) chương trình sẽ ngầm hiểu hàm main() trả lại một giá trị nguyên nhưng vì không có nên khi dịch chương trình ta sẽ gặp lời cảnh báo "Cần có giá trị trả lại cho hàm" (một lời cảnh báo không phải là lỗi, chương trình vẫn chạy bình thường). Để tránh bị quấy rầy về những lời cảnh báo "không mời" này chúng ta có thể đặt thêm câu lệnh return 0; (nếu không khai báo void main()) hoặc khai báo kiểu hàm là void main() và đặt câu lệnh return vào cuối hàm.

Chú ý:

- Danh sách đối trong khai báo hàm có thể chứa hoặc không chứa tên đối, thông thường ta chỉ khai báo kiểu đối chứ không cần khai báo tên đối, trong khi ở dòng đầu tiên của định nghĩa hàm phải có tên đối đầy đủ;
- Cuối khai báo hàm phải có dấu chấm phẩy (;), trong khi cuối dòng đầu tiên của định nghĩa hàm không có dấu chấm phẩy;
- Hàm có thể không có đối (danh sách đối rỗng), tuy nhiên cặp dấu ngoặc sau tên hàm vẫn phải được viết. Ví dụ clrscr(), lamtho(), vietgiaotrinh(), ... ;
- Một hàm có thể không cần phải khai báo nếu nó được định nghĩa trước khi có hàm nào đó gọi đến nó. Ví dụ có thể viết hàm main() trước (trong văn bản chương trình), rồi sau đó mới viết đến các hàm "con". Do trong hàm main() chắc chắn sẽ gọi đến hàm con này nên danh sách của chúng phải được khai báo trước hàm main(). Trường hợp ngược lại nếu các hàm con được viết (định nghĩa) trước thì không cần phải khai báo chúng nữa (vì trong định nghĩa đã hàm ý khai báo). Nguyên tắc này áp dụng cho hai hàm A, B bất kỳ chứ không riêng cho hàm main(), nghĩa là nếu B gọi đến A thì trước đó A phải được định nghĩa hoặc ít nhất cũng có dòng khai báo về A.

2. Lời gọi Hàm

Lời gọi hàm được phép xuất hiện trong bất kỳ biểu thức, câu lệnh của hàm khác ... Nếu lời gọi hàm lại nằm trong chính bản thân hàm đó thì ta gọi là đệ quy. Để gọi hàm ta chỉ cần viết tên hàm và danh sách các giá trị cụ thể truyền cho các đối đặt trong cặp dấu ngoặc tròn () .

Cú pháp

Tên_hàm(danh sách tham đối thực sự) ;

- Trong đó:
 - Danh sách tham đối thực sự còn gọi là danh sách giá trị gồm các giá trị cụ thể để gán lần lượt cho các đối hình thức của hàm. Khi hàm được gọi thực hiện thì tất cả những vị trí xuất hiện của đối hình thức sẽ được gán cho giá trị cụ thể của đối thực sự tương ứng trong danh sách, sau đó hàm tiến hành thực hiện các câu lệnh của hàm (để tính kết quả);
 - Danh sách tham đối thực sự truyền cho tham đối hình thức có số lượng bằng với số lượng đối trong hàm và được truyền cho đối theo thứ tự tương ứng. Các tham đối thực sự có thể là các hằng, các biến hoặc biểu thức. Biến trong giá trị có thể trùng với tên đối. Ví dụ ta có hàm in n lần kí tự c với tên hàm inkitu(int n, char c); và lời gọi hàm inkitu(12, 'A'); thì n và c là các đối hình thức, 12 và 'A' là các đối thực sự hoặc giá trị. Các đối hình thức n và c sẽ lần lượt được gán bằng các giá trị tương ứng là 12 và 'A' trước khi tiến hành các câu lệnh trong phần thân hàm. Giả sử hàm in kí tự được khai báo lại thành inkitu(char c, int n); thì lời gọi hàm cũng phải được thay lại thành inkitu('A', 12);
 - Các giá trị tương ứng được truyền cho đối phải có kiểu cùng với kiểu đối (hoặc C++ có thể tự động chuyển kiểu được về kiểu của đối);
 - Khi một hàm được gọi, nơi gọi tạm thời chuyển điều khiển đến thực hiện dòng lệnh đầu tiên trong hàm được gọi. Sau khi kết thúc thực hiện hàm, điều khiển lại được trả về thực hiện tiếp câu lệnh sau lệnh gọi hàm của nơi gọi.

- Ví dụ: Giả sử ta cần tính giá trị của biểu thức $2x^3 - 5x^2 - 4x + 1$, thay cho việc tính trực tiếp x^3 và x^2 , ta có thể gọi hàm `luythua()` trong ví dụ trên để tính các giá trị này bằng cách gọi nó trong hàm `main()` như sau:

```
#include <iostream.h>
#include <iomanip.h>
void xmh(int);
double luythua(float x, int n);
main()                                // tính giá trị  $2x^3 - 5x^2 - 4x + 1$ 
{
    float x ;                          // tên biến có thể trùng với đối của hàm
    double f ;                          // để lưu kết quả
    printf("x = ");
    scanf("%f" ,&x ) ;
    f = 2*luythua(x,3) - 5*luythua(x,2) - 4*x + 1;
    xmh(100);                            // xoá thật sạch màn hình 100 lần
    printf("%d",f) ;
    getch();
    return 0;
}
double luythua(float x, int n) // trả lại giá trị  $x^n$ 
{
    int i ;                              // biến chỉ số
    double kq = 1 ;                      // để lưu kết quả
    for (i=1; i<=n; i++) kq *= x ;
    return kq;
}
void xmh(int n) // xoá màn hình n lần
{
    int i;
    for (i=1; i<=n; i++) clrscr();
    return ;
}
```

Qua ví dụ này ta thấy lợi ích của lập trình cấu trúc, chương trình trở nên gọn hơn, chẳng hạn hàm `luythua()` chỉ được viết một lần nhưng có thể sử dụng nó nhiều lần (2 lần trong ví dụ này) chỉ bằng một câu lệnh gọi đơn giản cho mỗi lần sử dụng thay vì phải viết lại nhiều lần đoạn lệnh tính lũy thừa.

3. Hàm với đối mặc định

- Mục này và mục sau chúng ta bàn đến một vài mở rộng thiết thực của C/C++ đối với C có liên quan đến hàm, đó là hàm với đối mặc định và cách tạo, sử dụng các hàm có chung tên gọi. Một mở rộng quan trọng khác là cách truyền đối theo tham chiếu sẽ được bàn chung trong mục truyền tham đối thực sự cho hàm.
- Trong phần trước chúng ta đã khẳng định số lượng tham đối thực sự phải bằng số lượng tham đối hình thức khi gọi hàm. Tuy nhiên, trong thực tế rất nhiều lần hàm được gọi với các giá trị của một số tham đối hình thức được lặp đi lặp lại. Trong trường hợp như vậy lúc nào cũng phải viết một danh sách dài các tham đối thực sự giống nhau cho mỗi lần gọi là một công việc không mấy thú vị. Từ thực tế đó C++ đưa ra một cú pháp mới về hàm sao cho một danh sách tham đối thực sự trong lời gọi không nhất thiết phải viết đầy

đủ nếu một số trong chúng đã có sẵn những giá trị định trước.

Cú pháp

<kiểu hàm> <tên hàm>(đ1, ..., đn, đmđ1 = gt1, ..., đmđm = gtm

- Trong đó:
- Các đối đ1, ..., đn và đối mặc định đmđ1, ..., đmđm đều được khai báo như cũ nghĩa là gồm có kiểu đối và tên đối;
- Riêng các đối mặc định đmđ1, ..., đmđm có gán thêm các giá trị mặc định gt1, ..., gtm. Một lời gọi bất kỳ khi gọi đến hàm này đều phải có đầy đủ các tham đối thực sự ứng với các đ1, ..., đm nhưng có thể có hoặc không các tham đối thực sự ứng với các đối mặc định đmđ1, ..., đmđm. Nếu tham đối nào không có tham đối thực sự thì nó sẽ được tự động gán giá trị mặc định đã khai báo.
- Ví dụ 4.5: Xét hàm `xmh(int n = 100)`, trong đó `n` mặc định là 100, nghĩa là nếu gọi `xmh(99)` thì màn hình được xoá 99 lần, còn nếu gọi `xmh(100)` hoặc gọn hơn `xmh()` thì chương trình sẽ xoá màn hình 100 lần. Tương tự, xét hàm `int luythua(float x, int n = 2)`; Hàm này có một tham đối mặc định là số mũ `n`, nếu lời gọi hàm bỏ qua số mũ này thì chương trình hiểu là tính bình phương của `x` (`n = 2`). Ví dụ lời gọi `luythua(4, 3)` được hiểu là 4^3 , còn `luythua(2)` được hiểu là 2^2 .
- Hàm tính tổng 4 số nguyên: `int tong(int m, int n, int i = 0; int j = 0)`; khi đó có thể tính tổng của 5, 2, 3, 7 bằng lời gọi hàm `tong(5,2,3,7)` hoặc có thể chỉ tính tổng 3 số 4, 2, 1 bằng lời gọi `tong(4,2,1)` hoặc cũng có thể gọi `tong(6,4)` chỉ để tính tổng của 2 số 6 và 4.
- Chú ý: Các đối ngầm định phải được khai báo liên tục và xuất hiện cuối cùng trong danh sách đối. Ví dụ:

```
int tong(int x, int y=2, int z, int t=1);           // sai vì các đối mặc định không liên tục
void xoa(int x=0, int y)                          // sai vì đối mặc định không ở cuối
```

4. Khai báo hàm trùng tên

- Hàm trùng tên hay còn gọi là hàm chồng (đề). Đây là một kỹ thuật cho phép sử dụng cùng một tên gọi cho các hàm "giống nhau" (cùng mục đích) nhưng xử lý trên các kiểu dữ liệu khác nhau hoặc trên số lượng dữ liệu khác nhau. Ví dụ hàm sau tìm số lớn nhất trong 2 số nguyên:

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

- Nếu đặt `c = max(3, 5)` ta sẽ có `c = 5`.
- Tuy nhiên cũng tương tự như vậy nếu đặt `c = max(3.0, 5.0)` chương trình sẽ bị lỗi vì các giá trị (float) không phù hợp về kiểu (int) của đối trong hàm `max`.
- Trong trường hợp như vậy chúng ta phải viết hàm mới để tính `max` của 2 số thực. Mục đích, cách làm việc của hàm này hoàn toàn giống hàm trước, tuy nhiên trong C và các ngôn ngữ lập trình khác chúng ta buộc phải sử dụng một tên mới cho hàm "mới" này. Ví dụ:

```
float fmax(float a, float b) { return (a > b) ? a : b ; }
```

Tương tự để thuận tiện ta sẽ viết thêm các hàm:

```
char cmax(char a, char b) { return (a > b) ? a : b ; }
```

```
long lmax(long a, long b) { return (a > b) ? a : b ; }
```

```
double dmax(double a, double b) { return (a > b) ? a : b ; }
```

Tóm lại ta sẽ có 5 hàm: max, cmax, fmax, lmax, dmax, việc sử dụng tên như vậy sẽ gây bất lợi khi cần gọi hàm.

- C++ cho phép ta có thể khai báo và định nghĩa cả 5 hàm trên với cùng 1 tên gọi ví dụ là max chẳng hạn. Khi đó ta có 5 hàm:

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

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

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

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

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

Và lời gọi hàm bất kỳ dạng nào như max(3,5), max(3.0,5), max('O', 'K') đều được đáp ứng. Chúng ta có thể đặt ra vấn đề: với cả 5 hàm cùng tên như vậy, chương trình gọi đến hàm nào. Vấn đề được giải quyết dễ dàng vì chương trình sẽ dựa vào kiểu của các đối khi gọi để quyết định chạy hàm nào. Ví dụ lời gọi max(3,5) có 2 đối đều là kiểu nguyên nên chương trình sẽ gọi hàm 1, lời gọi max(3.0,5) hướng đến hàm số 2 và tương tự chương trình sẽ chạy hàm số 3 khi gặp lời gọi max('O','K').

- Một đặc điểm của các hàm trùng tên đó là trong danh sách đối của chúng phải có ít nhất một cặp đối nào đó khác kiểu nhau. Một đặc trưng khác để phân biệt thông qua các đối đó là số lượng đối trong các hàm phải khác nhau (nếu kiểu của chúng là giống nhau). Ví dụ việc vẽ các hình: thẳng, tam giác, vuông, chữ nhật trên màn hình là giống nhau, chúng chỉ phụ thuộc vào số lượng các điểm nối và tọa độ của chúng. Do vậy ta có thể khai báo và định nghĩa 4 hàm vẽ nói trên với cùng chung tên gọi. Chẳng hạn:

```
void ve(Diem A, Diem B) ; // vẽ đường thẳng AB
```

```
void ve(Diem A, Diem B, Diem C) ; // vẽ tam giác ABC
```

```
void ve(Diem A, Diem B, Diem C, Diem D) ; // vẽ tứ giác ABCD
```

Trong ví dụ trên ta giả thiết Diem là một kiểu dữ liệu lưu tọa độ của các điểm trên màn hình. Hàm ve(Diem A, Diem B, Diem C, Diem D) sẽ vẽ hình vuông, chữ nhật, thoi, bình hành hay hình thang phụ thuộc vào tọa độ của 4 điểm ABCD, nói chung nó được sử dụng để vẽ một tứ giác bất kỳ.

- Tóm lại nhiều hàm có thể được định nghĩa chồng (với cùng tên gọi giống nhau) nếu chúng thoả các điều kiện sau:
 - Số lượng các tham đối trong hàm là khác nhau, hoặc
 - Kiểu của tham đối trong hàm là khác nhau.

5. Truyền tham số

Có 3 cách truyền tham đối thực sự cho các tham đối hình thức trong lời gọi hàm. Trong đó cách ta đã dùng cho đến thời điểm hiện nay được gọi là truyền theo tham trị, tức các đối hình thức sẽ nhận các giá trị cụ thể từ lời gọi hàm và tiến hành tính toán rồi trả lại giá trị. Để dễ hiểu các cách truyền đối chúng ta sẽ xem qua cách thức chương trình thực hiện với các đối khi thực hiện hàm.

1. Truyền theo tham trị

- Ta xét lại ví dụ hàm lũy thừa(float x, int n) tính x^n . Giả sử trong chương trình chính ta có các biến a, b, f đang chứa các giá trị a = 2, b = 3, và f chưa có giá trị. Để tính a^b và gán giá trị tính được cho f, ta có thể gọi f = lũythua(a,b). Khi gặp lời gọi này, chương trình sẽ tổ chức như sau:
 - Tạo 2 biến mới (tức 2 ô nhớ trong bộ nhớ) có tên x và n. Gán nội dung các ô nhớ này bằng các giá trị trong lời gọi, tức gán 2 (a) cho x và 3 (b) cho n;
 - Tối ưu phân khai báo (của hàm), chương trình tạo thêm các ô nhớ mang tên kq và i;
 - Tiến hành tính toán (gán lại kết quả cho kq);
 - Cuối cùng lấy kết quả trong kq gán cho ô nhớ f (là ô nhớ có sẵn đã được khai báo trước, nằm bên ngoài hàm);
 - Kết thúc hàm quay về chương trình gọi. Do hàm lũythua đã hoàn thành xong việc tính toán nên các ô nhớ được tạo ra trong khi thực hiện hàm (x, n, kq, i) sẽ được xóa khỏi bộ nhớ. Kết quả tính toán được lưu giữ trong ô nhớ f (không bị xóa vì không liên quan gì đến hàm)
- Trên đây là truyền đối theo cách thông thường. Vấn đề đặt ra là giả sử ngoài việc tính f, ta còn muốn thay đổi các giá trị của các ô nhớ a, b (khi truyền nó cho hàm) thì có thể thực hiện được không? Để giải quyết bài toán này ta cần theo một kỹ thuật khác, nhờ vào vai trò của biến con trỏ và tham chiếu.

2. Truyền theo dẫn trỏ

- Xét ví dụ trao đổi giá trị của 2 biến. Đây là một yêu cầu nhỏ nhưng được gặp nhiều lần trong chương trình, ví dụ để sắp xếp một danh sách. Do vậy cần viết một hàm để thực hiện yêu cầu trên. Hàm không trả kết quả. Do các biến cần trao đổi là chưa được biết trước tại thời điểm viết hàm, nên ta phải đưa chúng vào hàm như các tham đối, tức hàm có hai tham đối x, y đại diện cho các biến sẽ thay đổi giá trị sau này.
- Từ một vài nhận xét trên, theo thông thường hàm trao đổi sẽ được viết như sau:

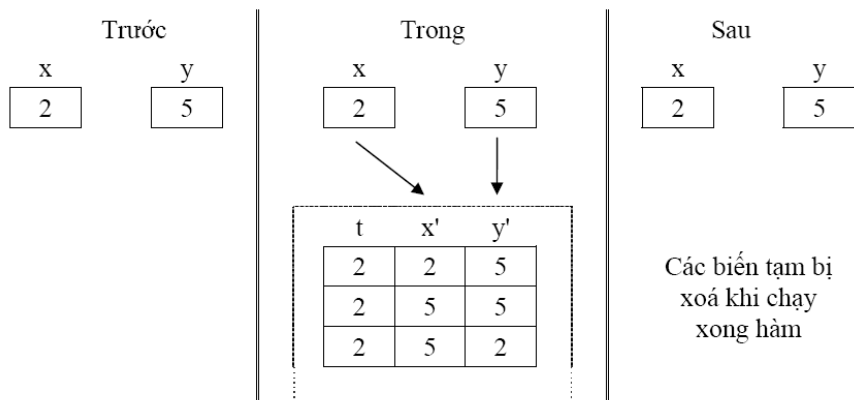
```
void doi_cho(int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
}
```

Giả sử trong chương trình chính ta có 2 biến x, y chứa các giá trị lần lượt là 2, 5. Ta cần đổi nội dung 2 biến này sao cho x = 5 còn y = 2 bằng cách gọi đến hàm doi_cho(x, y).

```
int main()
{
    int x = 2;
    int y = 5;
    doi_cho(x, y);
    printf("%d, %d", x, y);           //      2, 5 (x, y vẫn không đổi)
    getch();
    return 0;
}
```

Thực sự sau khi chạy xong chương trình ta thấy giá trị của x và y vẫn không thay đổi !?.

- Như đã giải thích trong mục trên (gọi hàm luythua), việc đầu tiên khi chương trình thực hiện một hàm là tạo ra các biến mới (các ô nhớ mới, độc lập với các ô nhớ x, y đã có sẵn) tương ứng với các tham đối, trong trường hợp này cũng có tên là x, y và gán nội dung của x, y (ngoài hàm) cho x, y (mới). Và việc cuối cùng của chương trình sau khi thực hiện xong hàm là xoá các biến mới này. Do vậy nội dung của các biến mới thực sự là có thay đổi, nhưng không ảnh hưởng gì đến các biến x, y cũ. Hình vẽ dưới đây minh hoạ cách làm việc của hàm swap, trước, trong và sau khi gọi hàm.



Hình 4.1 - Sự thay đổi giá trị của tham số thực

- Như vậy hàm doi_cho cần được viết lại sao cho việc thay đổi giá trị không thực hiện trên các biến tạm mà phải thực sự thực hiện trên các biến ngoài. Muốn vậy thay vì truyền giá trị của các biến ngoài cho đối, bây giờ ta sẽ truyền địa chỉ của nó cho đối, và các thay đổi sẽ phải thực hiện trên nội dung của các địa chỉ này. Đó chính là lý do ta phải sử dụng con trỏ để làm tham đối thay cho biến thường. Cụ thể hàm swap được viết lại như sau:

```
void doi_cho(int *p, int *q)
{
    int t;           // khai báo biến tạm t
    t = *p;         // đặt giá trị của t bằng nội dung nơi p trỏ tới
    *p = *q;        // thay nội dung nơi p trỏ bằng nội dung nơi q trỏ
    *q = t;         // thay nội dung nơi q trỏ tới bằng nội dung của t
}
```

Với cách tổ chức hàm như vậy rõ ràng nếu ta cho p trỏ tới biến x và q trỏ tới biến y thì hàm doi_cho sẽ thực sự làm thay đổi nội dung của x, y chứ không phải của p, q. Từ đó lời gọi hàm sẽ là doi_cho(&x, &y) (tức truyền địa chỉ của x cho p, p trỏ tới x và tương tự q trỏ tới y).

- Như vậy có thể tóm tắt 3 đặc trưng để viết một hàm làm thay đổi giá trị biến ngoài như sau:
 - Đối của hàm phải là con trỏ (ví dụ int *p);
 - Các thao tác liên quan đến đối này (trong thân hàm) phải thực hiện tại nơi nó trỏ đến (ví dụ *p = ...);
 - Lời gọi hàm phải chuyển địa chỉ cho p (ví dụ &x).
- Ngoài hàm doi_cho đã trình bày, ở đây ta đưa thêm ví dụ để thấy sự cần thiết phải có hàm

cho phép thay đổi biến ngoài. Ví dụ hàm giải phương trình bậc 2 với yêu cầu:

- Cho trước 3 số a, b, c như 3 hệ số của phương trình, cần tìm 2 nghiệm x1, x2 của nó.
- Không thể lấy giá trị trả lại của hàm để làm nghiệm vì giá trị trả lại chỉ có 1 trong khi ta cần đến 2 nghiệm.
- Do vậy ta cần khai báo 2 biến "ngoài" trong chương trình để chứa các nghiệm, và hàm phải làm thay đổi 2 biến này (tức chứa giá trị nghiệm giải được).
- Như vậy hàm được viết cần phải có 5 đối, trong đó 3 đối a, b, c đại diện cho các hệ số, không thay đổi và 2 biến x1, x2 đại diện cho nghiệm, 2 đối này phải được khai báo dạng con trỏ.
- Ngoài ra, phương trình có thể vô nghiệm, 1 nghiệm hoặc 2 nghiệm do vậy hàm sẽ trả lại giá trị là số nghiệm của phương trình, trong trường hợp 1 nghiệm (nghiệm kép), giá trị nghiệm sẽ được cho vào x1.

```
int gptb2(float a, float b, float c, float *p, float *q)
{
    float d ;           // để chứa Δ
    d = (b*b) - 4*a*c ;
    if (d < 0) return 0 ;
    else
        if (d == 0)
            { *p = -b/(2*a) ; return 1 ; }
        else
            {
                *p = (-b + sqrt(d))/(2*a) ;
                *q = (-b - sqrt(d))/(2*a) ;
                return 2 ;
            }
}
```

- Một ví dụ của lời gọi hàm trong chương trình chính như sau:

```
main()
{
    float a, b, c ;           // các hệ số
    float x1, x2 ;           // các nghiệm
    printf("Nhập hệ số: ");
    scanf("%f %f %f",&a, &b, &c);
    switch (gptb2(a, b, c, &x1, &x2))
    {
        case 0:
            printf("Phương trình vô nghiệm");
            break;
        case 1:
            printf("Phương trình có nghiệm kép x = %f", x1);
            break;
        case 2:
            printf("Phương trình có 2 nghiệm phân biệt:\n");
            printf("x1 = %f , và x2 = %f", x1, x2);
            break;
    }
}
```

}

- Trên đây chúng ta đã trình bày cách xây dựng các hàm cho phép thay đổi giá trị của biến ngoài. Một đặc trưng dễ nhận thấy là cách viết hàm tương đối phức tạp. Do vậy C++ đã phát triển một cách viết khác dựa trên đối tham chiếu và việc truyền đối cho hàm được gọi là truyền theo tham chiếu.

3. Truyền theo tham chiếu

- Một biến có thể được gán cho một bí danh mới, và khi đó chỗ nào xuất hiện biến thì cũng tương đương như dùng bí danh và ngược lại. Một bí danh như vậy được gọi là một biến tham chiếu, ý nghĩa thực tế của nó là cho phép "tham chiếu" tới một biến khác cùng kiểu của nó, tức sử dụng biến khác nhưng bằng tên của biến tham chiếu.
- Giống khai báo biến bình thường, tuy nhiên trước tên biến ta thêm dấu & (&). Có thể tạm phân biến thành 3 loại: biến thường với tên thường, biến con trỏ với dấu * trước tên và biến tham chiếu với dấu &.

<kiểu biến> &<tên biến tham chiếu> = <tên biến được tham chiếu>;

- Cú pháp khai báo này cho phép ta tạo ra một biến tham chiếu mới và cho nó tham chiếu đến biến được tham chiếu (cùng kiểu và phải được khai báo từ trước). Khi đó biến tham chiếu còn được gọi là bí danh của biến được tham chiếu. Chú ý không có cú pháp khai báo chỉ tên biến tham chiếu mà không kèm theo khởi tạo. Ví dụ:

```
int hung, dung ;           // khai báo các biến nguyên hung, dung
int &ti = hung;           // khai báo biến tham chiếu ti, teo tham chiếu đến
int &teo = dung;          // hung dung. ti, teo là bí danh của hung, dung
```

Từ vị trí này trở đi việc sử dụng các tên hung, ti hoặc dung, teo là như nhau. Ví dụ:

```
hung = 2 ;
ti ++;                       // tương đương hung ++;
printf("%d, %d",hung, ti);   // 3 3
teo = ti + hung ;           // tương đương dung = hung + hung
dung ++ ;                   // tương đương teo ++
printf("%d, %d",dung,teo);   // 7 7
```

- Vậy sử dụng thêm biến tham chiếu để làm gì ?
 - Cách tổ chức bên trong của một biến tham chiếu khác với biến thường ở chỗ nội dung của nó là địa chỉ của biến mà nó đại diện (giống biến con trỏ), ví dụ câu lệnh

```
printf("%d", teo) ;           // 7
```

in ra giá trị 7 nhưng thực chất đây không phải là nội dung của biến teo, nội dung của teo là địa chỉ của dung, khi cần in teo, chương trình sẽ tham chiếu đến dung và in ra nội dung của dung (7). Các hoạt động khác trên teo cũng vậy (ví dụ teo++), thực chất là tăng một đơn vị nội dung của dung (chứ không phải của teo). Từ cách tổ chức của biến tham chiếu ta thấy chúng giống con trỏ nhưng thuận lợi hơn ở chỗ khi truy cập đến giá trị của biến được tham chiếu (dung) ta chỉ cần ghi tên biến tham chiếu (teo) chứ không cần thêm toán tử (*) ở trước như trường hợp dùng con trỏ. Điểm khác biệt này có ích khi được sử dụng để truyền đối cho các

hàm với mục đích làm thay đổi nội dung của biến ngoài.

- Chú ý:
 - Biến tham chiếu phải được khởi tạo khi khai báo;
 - Tuy giống con trỏ nhưng không dùng được các phép toán con trỏ cho biến tham chiếu. Nói chung chỉ nên dùng trong truyền đối cho hàm.
- Một hàm viết dưới dạng đối tham chiếu sẽ đơn giản hơn rất nhiều so với đối con trỏ và giống với cách viết bình thường (truyền theo tham trị), trong đó chỉ có một khác biệt đó là các đối khai báo dưới dạng tham chiếu.
- Để so sánh 2 cách sử dụng ta nhắc lại các điểm khi viết hàm theo con trỏ phải chú ý đến, đó là:
 - Đối của hàm phải là con trỏ (ví dụ `int *p`);
 - Các thao tác liên quan đến đối này trong thân hàm phải thực hiện tại nơi nó trỏ đến (ví dụ `*p = ...`);
 - Lời gọi hàm phải chuyển địa chỉ cho `p` (ví dụ `&x`).
- Hãy so sánh với đối tham chiếu, cụ thể:
 - Đối của hàm phải là tham chiếu (ví dụ `int &p`);
 - Các thao tác liên quan đến đối này phải thực hiện tại nơi nó trỏ đến, tức địa chỉ cần thao tác. Vì một thao tác trên biến tham chiếu thực chất là thao tác trên biến được nó tham chiếu nên trong hàm chỉ cần viết `p` trong mọi thao tác (thay vì `*p` như trong con trỏ);
 - Lời gọi hàm phải chuyển địa chỉ cho `p`. Vì bản thân `p` khi tham chiếu đến biến nào thì sẽ chứa địa chỉ của biến đó, do đó lời gọi hàm chỉ cần ghi tên biến, ví dụ `x` (thay vì `&x` như đối với dẫn trỏ).
- Tóm lại, đối với hàm viết theo tham chiếu chỉ thay đổi ở đối (là các tham chiếu) còn lại mọi nơi khác đều viết đơn giản như cách viết truyền theo tham trị.
- Ví dụ: Đổi giá trị 2 biến

```
void doi_cho(int &x, int &y)
{
    int t;
    t = x;
    x = y;
    y = t;
}
```

và lời gọi hàm cũng đơn giản như trong truyền đối theo tham trị. Ví dụ:

```
int a = 5, b = 3;
doi_cho(a, b);
printf("%d, %d", a, b);
```

Bảng dưới đây minh họa tóm tắt 3 cách viết hàm thông qua ví dụ đổi biến ở trên

	Tham trị	Tham chiếu	Dẫn trở
Khai báo đổi	<code>void doi_cho(int x, int y)</code>	<code>void doi_cho(int &x, int &y)</code>	<code>void doi_cho(int *x, int *y)</code>
Câu lệnh	<code>t=x; x=y; y=t</code>	<code>t=x; x=y; y=t</code>	<code>t=*x; *x=*y; *y=t</code>
Lời gọi	<code>doi_cho(a,b)</code>	<code>doi_cho(a,b)</code>	<code>doi_cho(&a, &b)</code>
Tác dụng	a,b không thay đổi	a,b có thay đổi	a,b có thay đổi

Bảng 4.1 - So sánh các cách truyền tham số

6. Hàm và mảng

1. Truyền mảng 1 chiều cho hàm

- Thông thường chúng ta hay xây dựng các hàm làm việc trên mảng như vectơ hay ma trận các phần tử. Khi đó tham đối thực sự của hàm sẽ là các mảng dữ liệu này. Trong trường hợp này ta có 2 cách khai báo đối.

- Cách thứ nhất đối được khai báo bình thường như khai báo biến mảng nhưng không cần có số phần tử kèm theo, ví dụ:

```
int x[];
float x[];
```

- Cách thứ hai khai báo đối như một con trỏ kiểu phần tử mảng, ví dụ:

```
int *p;
float *p
```

- Trong lời gọi hàm tên mảng a (tham số thực) sẽ được viết vào danh sách tham đối thực sự, vì a là địa chỉ của phần tử đầu tiên của mảng a, nên khi hàm được gọi địa chỉ này sẽ gán cho con trỏ p. Vì vậy giá trị của phần tử thứ i của a có thể được truy cập bởi x[i] (theo khai báo 1) hoặc *(p+i) (theo khai báo 2) và nó cũng có thể được thay đổi thực sự (do đây cũng là cách truyền theo dẫn trở). Sau đây là ví dụ đơn giản, nhập và in vectơ, minh họa cho cả 2 kiểu khai báo đối.

- Ví dụ: Hàm nhập và in giá trị 1 vectơ

```
#include <stdio.h>
#include <conio.h>
void nhap(int x[], int n) // n: số phần tử
{
    int i;
    for (i=0; i<n; i++)
        scanf("%d",&x[i]);
}
void in(int *p, int n)
{
    int i;
    for (i=0; i<n; i++)
```

```

        printf("%d", *(p+i));
    }
    int main()
    {
        int a[10];           // mảng a chứa tối đa 10 phần tử
        nhap(a,7);          // vào 7 phần tử đầu tiên cho a
        in(a,7);            // ra 3 phần tử đầu tiên của a
        getch();
        return 0;
    }

```

2. Truyền mảng 2 chiều cho hàm

- Đối với mảng 2 chiều khai báo đối cũng như lời gọi là phức tạp hơn nhiều so với mảng 1 chiều. Ta có hai cách khai báo đối như sau:

a. Khai báo theo đúng bản chất của mảng 2 chiều float x[m][n]

- Với khai báo x[m][n] có thể hiểu x là mảng 1 chiều m phần tử, mỗi phần tử của nó có kiểu float[n]. Từ đó, đối được khai báo như một mảng hình thức 1 chiều (không cần số phần tử - ở đây là số dòng) của kiểu float[n]. Tức có thể khai báo như sau:

```

float x[ ][n]; // mảng với số phần tử không định trước, mỗi phần tử là n số
float (*x)[n]; // một con trỏ, có kiểu là mảng n số (float[n])

```

Để truy nhập đến phần tử thứ i, j ta vẫn sử dụng cú pháp x[i][j]. Tên của mảng a (tham số thực) được viết bình thường trong lời gọi hàm. Nói chung theo cách khai báo này việc truy nhập là đơn giản nhưng phương pháp cũng có hạn chế đó là số cột của mảng truyền cho hàm phải cố định bằng n.

- Ví dụ: Tính tổng các số hạng trong ma trận

```

#include <stdio.h>
#include <conio.h>
float tong(float x[][10], int m, int n)
{
    float t = 0;
    int i, j;
    for (i=0; i<m; i++)
        for (j=0; j<n; j++) t += x[i][j];
    return t;
}
int main()
{
    float a[8][10], b[8][10];
    int i, j, ma, na, mb, nb;
    printf("nhap so dong, so cot ma tran a: ");
    scanf("%d%d",&ma,&na);
    for (i=0; i<ma; i++)
        for (j=0; j<na; j++)
        {
            printf("a[%d,%d]= ",i,j);
            scanf("%f",&a[i][j]);
        }
    printf("nhap so dong, so cot ma tran b: ");
    scanf("%d%d",&mb,&nb);

```

```

for (i=0; i<mb; i++)
    for (j=0; j<nb; j++)
    {
        printf("a[%d,%d]= ",i,j);
        scanf("%f",&b[i][j]);
    }
printf("%f\n",tong(a, ma, na));
printf("%f\n",tong(b, mb, nb));
getch();
return 0;
}

```

b. Xem mảng float x[m][n] thực sự là mảng một chiều float x[m*n]

- Khi đó sử dụng cách khai báo như trong mảng một chiều, đó là sử dụng con trỏ float *p để truy cập được đến từng phần tử của mảng. Cách này có hạn chế trong lời gọi: địa chỉ truyền cho hàm không phải là mảng a mà cần phải ép kiểu về (float*) (để phù hợp với p). Với cách này gọi k là thứ tự của phần tử a[i][j] trong mảng một chiều (m*n), ta có quan hệ giữa k, i, j như sau: $k = *(p + i*n + j)$, $i = k/n$ và $j = k \% n$ trong đó n là số cột của mảng truyền cho hàm. Điều này có nghĩa để truy cập đến a[i][j] ta có thể viết *(p+i*n+j), ngược lại biết chỉ số k có thể tính được dòng i, cột j của phần tử này. Ưu điểm của cách khai báo này là ta có thể truyền mảng với kích thước bất kỳ (số cột không cần định trước) cho hàm.
- Ví dụ: Tìm phần tử bé nhất của ma trận

```

#include <stdio.h>
#include <conio.h>
void minmt(float *x, int m, int n)
{
    float min = *x;
    int k, kmin;
    for (k=1; k<m*n; k++)
        if (min > x[k])
        {
            min = x[k];
            kmin = k;
        }
    printf("Gia tri min la: %f , tai dong %d, cot %d\n",min,kmin/n,kmin%n);
}
int main()
{
    float a[3][3];
    int i, j ;
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
        {
            printf("a[%d,%d]",i,j);
            scanf("%f",&a[i][j]);
        }
    minmt((float*)(a), 3, 3) ;
    getch();
    return 0;
}

```

3. Giá trị trả lại của hàm là mảng

- Không có cách nào để giá trị trả lại của một hàm là mảng. Tuy nhiên thực sự mỗi mảng cũng chính là một con trỏ, vì vậy việc hàm trả lại một con trỏ trỏ đến dãy dữ liệu kết quả là tương đương với việc trả lại mảng. Ngoài ra còn một cách dễ dùng hơn đối với mảng 2 chiều là mảng kết quả được trả lại vào trong tham đối của hàm (giống như nghiệm của phương trình bậc 2 được trả lại vào trong các tham đối). Ở đây chúng ta sẽ lần lượt xét 2 cách làm việc này.

Giá trị trả lại là con trỏ trỏ đến mảng kết quả

- Trước hết chúng ta xét ví dụ sau đây:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int* day1()
{
    int kq[3] = { 7, 5, 3 };
    return kq;
}
int* day2()
{
    int *kq;
    kq=(int*)malloc(3*sizeof(int));
    *kq = *(kq+1) = *(kq+2) = 0 ;
    return kq ;
}
int main()
{
    int *a, i;
    a = day1();
    for (i=0; i<3; i++)
        printf("%d\n",*(a+i));
    a = day2();
    for (i=0; i<3; i++)
        printf("%d\n",*(a+i));
    getch();
    return 0;
}
```

- Từ ví dụ trên ta thấy hai hàm trả giá trị đều tạo bên trong nó một mảng 3 số nguyên và trả lại địa chỉ mảng này cho con trỏ kết quả hàm. Tuy nhiên, chỉ có day2() là cho lại kết quả đúng. Tại sao ?
 - Xét mảng kq được khai báo và khởi tạo trong day1(), đây là một mảng cục bộ (được tạo bên trong hàm) như sau này chúng ta sẽ thấy, các loại biến "tạm thời" này (và cả các tham đối) chỉ tồn tại trong quá trình hàm hoạt động. Khi hàm kết thúc các biến này sẽ mất đi. Do vậy tuy hàm đã trả lại địa chỉ của kq trước khi nó kết thúc, thế nhưng sau khi hàm thực hiện xong, toàn bộ kq sẽ được xóa khỏi bộ nhớ và vì vậy con trỏ kết quả hàm đã trỏ đến vùng nhớ không còn các giá trị như kq đã có. Từ điều này việc sử dụng hàm trả lại con trỏ là phải hết sức cẩn thận. Muốn trả lại con trỏ cho hàm thì con trỏ này phải trỏ đến dãy dữ liệu nào sao cho nó không mất đi sau khi hàm kết thúc, hay nói khác hơn đó phải là những dãy dữ

liệu được khởi tạo bên ngoài hàm hoặc có thể sử dụng theo phương pháp trong hàm day2().

- Trong day2() một mảng kết quả 3 số cũng được tạo ra nhưng bằng cách xin cấp phát vùng nhớ. Vùng nhớ được cấp phát này sẽ vẫn còn tồn tại sau khi hàm kết thúc (nó chỉ bị xoá đi khi sử dụng toán tử delete). Do vậy hoạt động của day2() là chính xác.

Mảng cần trả lại được khai báo như một tham đối trong danh sách đối của hàm

- Tham đối này là một con trỏ nên hiển nhiên khi truyền mảng đã khai báo sẵn (để chứa kết quả) từ ngoài vào cho hàm thì mảng sẽ thực sự nhận được nội dung kết quả (tức có thay đổi trước và sau khi gọi hàm xem mục truyền tham đối thực sự theo dẫn trỏ). Để nắm được vấn đề này chúng ta xét ví dụ sau.
- Ví dụ: Cộng 2 vectơ, vectơ kết quả trả lại trong tham đối của hàm. So với ví dụ trước giá trị trả lại là void (không trả lại giá trị) còn danh sách đối có thêm con trỏ z để chứa kết quả.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void congvt(int *x, int *y, int *z, int n)
{
    for (int i=0; i<n; i++) z[i] = x[i] + y[i];
}
main()
{
    int i, n, a[10], b[10], c[10];
    printf("Nhap vao so phan tu: ");
    scanf("%d",&n);
    for (i=0; i<n; i++)
    {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
    }
    for (i=0; i<n; i++)
    {
        printf("b[%d]=",i);
        scanf("%d",&b[i]);
    }
    congvt(a, b, c, n);
    for (i=0; i<n; i++)
    {
        printf("c[%d]=%d\n",i,c[i]);
    }
    getch();
    return 0;
}
```

4. Đối và giá trị trả lại của hàm là chuỗi ký tự

- Giống các trường hợp đã xét với mảng 1 chiều, đối của các hàm chuỗi ký tự có thể khai báo dưới 2 dạng: mảng ký tự hoặc con trỏ ký tự. Giá trị trả lại luôn luôn là con trỏ ký tự. Ngoài ra hàm cũng có thể trả lại giá trị vào trong các đối con trỏ trong danh sách đối.

- Ví dụ sau đây dùng để tách họ, tên của một xâu họ và tên. Ví dụ gồm 3 hàm. Hàm họ trả lại xâu họ (con trỏ kí tự) với đối là xâu họ và tên được khai báo dạng mảng. Hàm tên trả lại xâu tên (con trỏ kí tự) với đối là xâu họ và tên được khai báo dạng con trỏ kí tự. Thực chất đối họ và tên trong hai hàm họ, tên có thể được khai báo theo cùng cách thức, ở đây chương trình muốn minh họa các cách khai báo đối khác nhau (đã đề cập đến trong phần đối mảng 1 chiều). Hàm thứ ba cũng trả lại họ, tên nhưng cho vào trong danh sách tham đối, do vậy hàm không trả lại giá trị (void). Để đơn giản ta qui ước xâu họ và tên không chứa các dấu cách đầu và cuối xâu, trong đó họ là dãy kí tự từ đầu cho đến khi gặp dấu cách đầu tiên và tên là dãy kí tự từ sau dấu cách cuối cùng đến kí tự cuối xâu.

- Ví dụ:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
char* ho(char hoten[])
{
    char* kq;
    int i=0;
    while (hoten[i] != 32) i++;
    kq=(char*)malloc(i*sizeof(char));
    strncpy(kq, hoten, i) ;
    return kq;
}
char* ten(char* hoten)
{
    char* kq = new char[10];
    int i=strlen(hoten);
    while (hoten[i] != 32) i--;
    kq=hoten+i+1;
    return kq;
}
void tachht(char* hoten, char* ho, char* ten)
{
    int i=0;
    while (hoten[i] != '\40') i++;
    strncpy(ho, hoten, i) ;
    i=strlen(hoten);
    while (hoten[i] != '\40') i--;
    strncpy(ten, hoten+i+1, strlen(hoten)-i-1) ;
}
int main()
{
    char ht[30], *h, *t ;
    printf("Ho va ten = ");
    gets(ht);
    h = ho(ht); t = ten(ht);
    printf("Ho:%s, Ten:%s.\n",h,t);
    tachht(ht, h, t);
    printf("Ho:%s, Ten:%s.\n",h,t);
    getch();
    return 0;
}
```

5. Đối là hằng con trỏ

- Theo phần truyền đối cho hàm ta đã biết để thay đổi biến ngoài đối tượng ứng phải được khai báo dưới dạng con trỏ. Tuy nhiên, trong nhiều trường hợp các biến ngoài không có nhu cầu thay đổi nhưng đối tượng ứng với nó vẫn phải khai báo dưới dạng con trỏ (ví dụ đối là mảng hoặc xâu kí tự). Điều này có khả năng do nhầm lẫn, các biến ngoài này sẽ bị thay đổi ngoài ý muốn. Trong trường hợp như vậy để cẩn thận, các đối con trỏ nếu không muốn thay đổi (chỉ lấy giá trị) cần được khai báo như là một hằng con trỏ bằng cách thêm trước khai báo kiểu của chúng từ khoá const. Từ khoá này khẳng định biến tuy là con trỏ nhưng nó là một hằng không thay đổi được giá trị. Nếu trong thân hàm ta cố tình thay đổi chúng thì chương trình sẽ báo lỗi. Ví dụ đối hoten trong cả 3 hàm ở trên có thể được khai báo dạng const char* hoten.
- Ví dụ: Đối là hằng con trỏ. In hoa một xâu kí tự

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
void inhoa(const char* s)
{
    char *t= new char[5];
    strcpy(t, s);
    printf("%s,%s",s,strupr(t));
}
int main()
{
    char *s = "abcde" ;
    inhoa(s);
    getch();
    return 0;
}
```

Trong ví dụ trên, nếustrupr(t) bằngstrupr(s) thì chương trình sẽ thông báo lỗi vì s được định nghĩa là hằng con trỏ trong khi hàmstrupr() lại làm thay đổi giá trị của nó, và điều này là không thể.

III. Con trỏ hàm

- Mặc dù một hàm không phải là một biến nhưng nó vẫn chiếm vị trí trong bộ nhớ và ta có thể gán vào vị trí của nó cho một loại biến con trỏ. Con trỏ này trỏ đến điểm xâm nhập vào hàm. Ta gọi đây là con trỏ hàm. Con trỏ hàm có thể sử dụng thay cho tên hàm và việc sử dụng con trỏ cho phép các hàm cũng được truyền như là tham số cho các hàm khác. Để hiểu được các con trỏ hàm làm việc như thế nào, ta cần hiểu một chút về cách biên dịch và gọi một hàm. Khi biên dịch hàm, trình biên dịch chuyển chương trình nguồn sang dạng mã máy và thiết lập một điểm xâm nhập vào hàm (chính là vị trí chỉ thị mã máy đầu tiên của hàm). Khi có lời gọi thực hiện hàm, máy tính sẽ thực hiện một chỉ thị call chuyển điều khiển đến điểm xâm nhập này. Trong trường hợp gọi hàm bằng tên hàm thì điểm xâm nhập này là trị tức thời (gần như là một hằng và không chứa biến nào cả), cách gọi hàm này gọi là cách gọi hàm trực tiếp. Trái lại, khi gọi hàm gián tiếp thông qua một biến trỏ thì biến trỏ đó phải trỏ tới chỉ thị mã máy đầu tiên của hàm đó. Cách gọi hàm thông qua biến trỏ hàm gọi là cách gọi hàm gián tiếp.

- Một hàm cũng giống như dữ liệu: có tên gọi, có địa chỉ lưu trong bộ nhớ và có thể truy cập đến hàm thông qua tên gọi hoặc địa chỉ của nó. Con trỏ hàm là một con trỏ chỉ đến địa chỉ của một hàm. Để truy cập (gọi) hàm thông qua địa chỉ chúng ta phải khai báo một con trỏ chứa địa chỉ này và sau đó gọi hàm bằng cách gọi tên con trỏ.

1. Khai báo

Cú pháp

```
<kiểu giá trị> (*tên biến hàm)(d/s tham đối);
<kiểu giá trị> (*tên biến hàm)(d/s tham đối) = <tên hàm>;
```

- Ta thấy cách khai báo con trỏ hàm cũng tương tự khai báo con trỏ biến (chỉ cần đặt dấu * trước tên), ngoài ra còn phải bao *tên hàm giữa cặp dấu ngoặc (). Ví dụ:

```
float (*f)(int);           // khai báo con trỏ hàm có tên là f trỏ đến hàm
                           // có một tham đối kiểu int và cho giá trị kiểu float.
void (*f)(float, int);    // con trỏ trỏ đến hàm với cặp đối (float, int).
```

- hoặc phức tạp hơn:

```
char* (*m[10])(int, char) // khai báo một mảng 10 con trỏ hàm trỏ đến
                           // các hàm có cặp tham đối (int, char), giá trị trả
                           // lại của các hàm này là chuỗi kí tự.
```

- Chú ý: phân biệt giữa 2 khai báo: float (*f)(int) và float *f(int). Cách khai báo trước là khai báo con trỏ hàm có tên là f. Cách khai báo sau có thể viết lại thành float* f(int) là khai báo hàm f với giá trị trả lại là một con trỏ float.

2. Sử dụng con trỏ hàm

- Con trỏ hàm thay tên hàm
- Để sử dụng con trỏ hàm ta phải gán nó với tên hàm cụ thể và sau đó bất kỳ nơi nào được phép xuất hiện tên hàm thì ta đều có thể thay nó bằng tên con trỏ. Ví dụ như các thao tác gọi hàm, đưa hàm vào làm tham đối hình thức cho một hàm khác ... Sau đây là các ví dụ minh họa.
- Ví dụ 4.16: Dùng tên con trỏ để gọi hàm

```
#include <stdio.h>
#include <conio.h>
float bphuong(float x)
{
    return x*x;
}
int main()
{
    float (*f)(float);
    f = bphuong;
    printf("Binh phuong cua 3.5 = :%f",f(3.5));
    getch();
    return 0;
}
```

- Con trỏ hàm làm tham đối
- Ví dụ 4.17: Tham đối của hàm ngoài các kiểu dữ liệu đã biết còn có thể là một hàm. Điều

này có tác dụng rất lớn trong các bài toán tính toán trên những đối tượng là hàm toán học như tìm nghiệm, tính tích phân của hàm trên một đoạn ... Hàm đóng vai trò tham đối sẽ được khai báo dưới dạng con trỏ hàm. Ví dụ sau đây trình bày hàm tìm nghiệm xấp xỉ của một hàm liên tục và đổi dấu trên đoạn $[a, b]$. Để hàm tìm nghiệm này sử dụng được trên nhiều hàm toán học khác nhau, trong hàm sẽ chứa một biến con trỏ hàm và hai cận a, b , cụ thể bằng khai báo `float timnghiem(float (*f)(float), float a, float b)`. Trong lời gọi hàm f sẽ được thay thế bằng tên hàm cụ thể cần tìm nghiệm.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define EPS 1.0e-6
float timnghiem(float (*f)(float), float a, float b);
float emux(float);
float logx(float);
int main()
{
    printf("Nghiem cua e mu x - 2 trên đoạn [0,1] = ");
    printf("%f\n",timnghiem(emux,0,1));
    printf("Nghiem cua loga(x) - 1 trên đoạn [2,3] = ");
    printf("%f\n",timnghiem(logx,0,1));
    getch();
    return 0;
}
float timnghiem(float (*f)(float), float a, float b)
{
    float c = (a+b)/2;
    while (fabs(a-b)>EPS && f(c)!=0)
    {
        if (f(a)*f(c)>0) a = c ; else b = c;
        c = (a+b)/2;
    }
    return c;
}
float emux(float x) { return (exp(x)-2); }
float logx(float x) { return (log(x)-1); }
```

3. Mảng con trỏ hàm

- Tương tự như biến bình thường các con trỏ hàm giống nhau có thể được gộp lại vào trong một mảng, trong khai báo ta chỉ cần thêm $[n]$ vào sau tên mảng với n là số lượng tối đa các con trỏ; ví dụ sau minh họa cách sử dụng này. Dùng mảng con trỏ để lập bảng giá trị cho các hàm : $x*x$, $\sin(x)$, $\cos(x)$, $\exp(x)$ và \sqrt{x} . Biến x chạy từ 1.0 đến 10.0 theo bước 0.5
- Ví dụ 4.18:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
double bp(double x)
{
    return x*x;
}
```

```

int main()
{
    int i,j;
    double x=1.0;
    double (*f[6])(double);
    f[1]=bp;           // hàm bình phương xây dựng ở trên
    f[2]=sin;         // hàm sin trong math.h
    f[3]=cos;         // hàm cos trong math.h
    f[4]=exp;         // hàm exp (e mũ) trong math.h
    f[5]=sqrt;        // hàm sqrt (căn bậc 2) trong math.h
    while (x<=10.0)  // Lập bảng
    {
        printf("\n");
        for (j=1;j<=5;++j)
            printf("%10.2f ",f[j](x));
        x+=0.5;
    }
    getch();
    return 0;
}

```

IV. Đệ qui

1. Khái niệm

- Một hàm được gọi là đệ qui nếu bên trong thân hàm có lệnh gọi đến chính nó. Khi hàm gọi đệ qui đến chính nó, thì mỗi lần gọi máy sẽ tạo ra một tập các biến cục bộ mới hoàn toàn độc lập với tập các biến cục bộ đã được tạo ra trong các lần gọi trước. Để minh họa chi tiết những điều trên, ta xét một ví dụ về tính giai thừa của số nguyên dương n . Theo định nghĩa giai thừa của một số nguyên n được ký hiệu là $n!$ và được tính như sau:

$$- \quad n! = 1 * 2 * 3 * \dots * (n-1) * n = (n-1)! * n \text{ (với } 0! = 1)$$

- Như vậy, để tính $n!$ ta thấy nếu $n=0$ thì $n!=1$ ngược lại thì $n!=n * (n-1)!$
- Ví dụ 4.19: Để minh họa chi tiết những điều trên, ta xét một ví dụ về tính giai thừa của số nguyên dương n . Khi không dùng phương pháp đệ qui hàm có thể được viết như sau :

```

long int gt1(int n) /* Tính n! v?i n>=0*/
{
    long int gtpHu=1;
    int i;
    for (i=1;i<=n;++i)
        gtpHu*=i;
    return gtpHu;
}

```

- Hàm tính $n!$ theo phương pháp đệ qui có thể được viết như sau :

```

long int gt2(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return(n*gt2(n-1));
}

```

```
}
```

- Và lời gọi hàm gt từ chương trình chính được thực hiện như sau:

```
int main()
{
    //printf("\n 3!=%d",gt1(3));
    printf("\n 3!=%d",gt2(3));
    getch();
    return 0;
}
```

- Lần gọi đầu tiên tới hàm gt2 được thực hiện từ hàm main(). Máy sẽ tạo ra một tập các biến tự động của hàm gt2. Tập này chỉ gồm các đối n. Ta gọi đối n được tạo ra lần thứ nhất là n thứ nhất. Giá trị của tham số thực (số 3) được gán cho n thứ nhất. Lúc này biến n trong thân hàm được xem là n thứ nhất. Do n thứ nhất có giá trị bằng 3 nên điều kiện trong toán tử if là sai và do đó máy sẽ lựa chọn câu lệnh else. Theo câu lệnh này, máy sẽ tính giá trị biểu thức: $n * gt2(n-1)$ (*). Để tính biểu thức trên, máy cần gọi chính hàm gt2 vì thế lần gọi thứ hai sẽ thực hiện. Máy sẽ tạo ra đối n mới, ta gọi đó là n thứ hai. Giá trị của n-1 ở đây lại là đối của hàm, được truyền cho hàm và hiểu là n thứ hai, do vậy n thứ hai có giá trị là 2. Bây giờ, do n thứ hai vẫn chưa thoả mãn điều kiện if nên máy lại tiếp tục tính biểu thức : $n * gt2(n-1)$ (**). Biểu thức trên lại gọi hàm gt2 lần thứ ba. Máy lại tạo ra đối n lần thứ ba và ở đây n thứ ba có giá trị bằng 1. Đối n=1 thứ ba lại được truyền cho hàm, lúc này điều kiện trong lệnh if được thoả mãn, máy đi thực hiện câu lệnh: $return 1=gt2(1)$ (***). Bắt đầu từ đây, máy sẽ thực hiện ba lần ra khỏi hàm gt2. Lần ra khỏi hàm thứ nhất ứng với lần vào thứ ba. Kết quả là đối n thứ ba được giải phóng, hàm gt2(1) cho giá trị là 1 và máy trở về xét giá trị biểu thức $n * gt2(1)$ đây là kết quả của (**) ở đây, n là n thứ hai và có giá trị bằng 2. Theo câu lệnh return, máy sẽ thực hiện lần ra khỏi hàm lần thứ hai, đối n thứ hai sẽ được giải phóng, kết quả là biểu thức trong (**) có giá trị là 2.1. Sau đó máy trở về biểu thức (*) lúc này là: $n * gt2(2)=n * 2 * 1$ n lại hiểu là thứ nhất, nó có giá trị bằng 3, do vậy giá trị của biểu thức trong (*) là $3 * 2 * 1 = 6$. Chính giá trị này được sử dụng trong câu lệnh printf của hàm main() nên kết quả in ra trên màn hình là: $3! = 6$.
- Từ ví dụ trên ta thấy hàm đệ qui có đặc điểm:
- Chương trình viết rất gọn,
- Việc thực hiện gọi đi gọi lại hàm rất nhiều lần phụ thuộc vào độ lớn của đầu vào. Chẳng hạn trong ví dụ trên hàm được gọi n lần, mỗi lần như vậy chương trình sẽ mất thời gian để lưu giữ các thông tin của hàm gọi trước khi chuyển điều khiển đến thực hiện hàm được gọi. Mặt khác các thông tin này được lưu trữ nhiều lần trong ngăn xếp sẽ dẫn đến tràn ngăn xếp nếu n lớn.
- Hàm đệ qui so với hàm có thể dùng vòng lặp thì đơn giản hơn, tuy nhiên với máy tính khi dùng hàm đệ qui sẽ dùng nhiều bộ nhớ trên ngăn xếp và có thể dẫn đến tràn ngăn xếp. Vì vậy khi gặp một bài toán mà có thể có cách giải lặp (không dùng đệ qui) thì ta nên dùng cách lặp này. Song vẫn tồn tại những bài toán chỉ có thể giải bằng đệ qui.

2. Lớp các bài toán giải được bằng đệ qui

- Phương pháp đệ qui thường được dùng để giải các bài toán có đặc điểm:
- Giải quyết được dễ dàng trong các trường hợp riêng gọi là trường hợp *suy biến* hay *cơ*

- sở, trong trường hợp này hàm được tính bình thường mà không cần gọi lại chính nó;
- Đối với trường hợp *tổng quát*, bài toán có thể giải được bằng bài toán cùng dạng nhưng với tham đối khác có kích thước nhỏ hơn tham đối ban đầu. Và sau một số bước hữu hạn biến đổi cùng dạng, bài toán đưa được về trường hợp suy biến.
 - Như vậy trong trường hợp tính $n!$ nếu $n = 0$ hàm cho ngay giá trị 1 mà không cần phải gọi lại chính nó, đây chính là trường hợp suy biến. Trường hợp $n > 0$ hàm sẽ gọi lại chính nó nhưng với n giảm 1 đơn vị. Việc gọi này được lặp lại cho đến khi $n = 0$.
 - Một lớp rất rộng của bài toán dạng này là các bài toán có thể định nghĩa được dưới dạng đệ quy như các bài toán lặp với số bước hữu hạn biết trước, các bài toán UCLN, tháp Hà Nội, ...
 - *Cấu trúc chung của hàm đệ quy*
 - Dạng thức chung của một chương trình đệ quy thường như sau:

```

if (trường hợp suy biến)
{
    trình bày cách giải // giả định đã có cách giải
}
else // trường hợp tổng quát
{
    gọi lại hàm với tham đối "bé" hơn
}

```

3. Các ví dụ

- Bài toán tìm ước số chung lớn nhất (UCLN): Cho hai số nguyên a và b , tìm USCLN của hai số theo giải thuật đệ quy. Bài toán có thể được định nghĩa dưới dạng đệ quy như sau:
- Nếu $b = 0$ thì $UCLN = a$;
- Nếu $b \neq 0$ thì $UCLN(a, b) = UCLN(b, a \bmod b)$.
- Từ đó ta có hàm đệ quy để tính UCLN của a và b như sau.
- Ví dụ 4.20:

```

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

```

- Số Fibonacci: Tính số hạng thứ n của dãy Fibonacci với giải thuật đệ quy. Dãy Fibonacci được định nghĩa như sau:
- $f(0) = f(1) = 1$;
- $f(n) = f(n-1) + f(n-2)$ với $\forall n \geq 2$.

Ví dụ 4.21:

```

long Fib(int n)
{
    long kq;

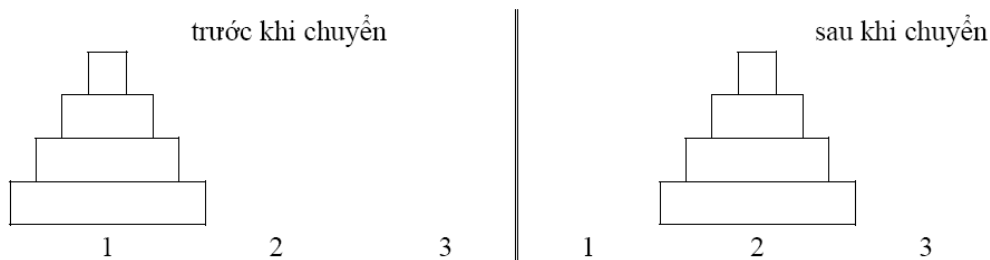
```

```

    if (n==0 || n==1) kq = 1;
    else kq = Fib(n-1) + Fib(n-2);
    return kq;
}

```

- Bài toán tháp Hà nội: Có n đĩa với kích thước to, nhỏ khác nhau. Ban đầu sắp xếp các đĩa theo trật tự kích thước vào một cọc sao cho đĩa nhỏ nhất nằm trên cùng, tức là tạo ra một dạng hình nón. Người chơi phải di chuyển toàn bộ số đĩa sang một cọc khác, tuân theo các quy tắc sau:
 - Một lần chỉ được di chuyển một đĩa;
 - Một đĩa chỉ có thể được đặt lên một đĩa lớn hơn (không nhất thiết hai đĩa này phải có kích thước liền kề, tức là đĩa nhỏ nhất có thể nằm trên đĩa lớn nhất).
- Hình ảnh mô tả bài toán được cho như hình 4.2.



Hình 4.2 - Bài toán tháp Hanoi

- Mô tả bài toán:
 - Đặt tên các cọc là A, B, C và qui ước A là Cọc Nguồn, B là Cọc Đích và C là Cọc Trung Gian;
 - Gọi n là tổng số đĩa;
 - Đánh số đĩa từ 1 (nhỏ nhất, trên cùng) đến n (lớn nhất, dưới cùng).
 - Thuật giải đệ qui: Để chuyển n đĩa từ cọc A sang cọc B với C là cọc trung gian:
 1. Chuyển $n-1$ đĩa từ A sang C. Chỉ còn lại đĩa # n trên cọc A;
 2. Chuyển đĩa # n từ A sang B;
 3. Chuyển $n-1$ đĩa từ C sang B cho chúng nằm trên đĩa # n
- Ví dụ 4.22:

```

#include <stdio.h>
#include <conio.h>
void chuyen(int sodia, char CotNguon, char CotDich, char CotTG)
{
    if (sodia>0)
    {
        chuyen(sodia-1, CotNguon, CotTG, CotDich);
        printf("Chuyen dia %d Tu %c -> %c\n",sodia,CotNguon,CotDich);
        chuyen(sodia-1, CotTG, CotDich, CotNguon);
    }
}
int main()
{
    char CotNguon,CotDich,CotTG;

```



```

int n;
printf("\n Hay nhap so dia: ");
scanf("%d",&n);
chuyen(n,'A','B','C');
getch();
}

```

- Kết quả thực hiện ví dụ 4.22 được cho như hình 4.3.

```

Hay nhap so dia: 4
Chuyen dia 1 Tu A -> C
Chuyen dia 2 Tu A -> B
Chuyen dia 1 Tu C -> B
Chuyen dia 3 Tu A -> C
Chuyen dia 1 Tu B -> A
Chuyen dia 2 Tu B -> C
Chuyen dia 1 Tu A -> C
Chuyen dia 4 Tu A -> B
Chuyen dia 1 Tu C -> B
Chuyen dia 2 Tu C -> A
Chuyen dia 1 Tu B -> A
Chuyen dia 3 Tu C -> B
Chuyen dia 1 Tu A -> C
Chuyen dia 2 Tu A -> B
Chuyen dia 1 Tu C -> B

```

Hình 4.3 - Kết quả thực hiện bài toán tháp Hà nội với n=4

V. Tóm tắt nội dung bài học

I. Tổ chức chương trình

1. Ví dụ
2. Cấu trúc chương trình
3. Hàm xây dựng sẵn

II. Hàm do người dùng định nghĩa

1. Khai báo và định nghĩa Hàm
2. Lời gọi Hàm
3. Hàm với đối mặc định
4. Khai báo hàm trùng tên
5. Truyền tham số
6. Hàm và mảng

III. Con trỏ hàm

1. Khai báo
2. Sử dụng con trỏ hàm
3. Mảng con trỏ hàm

IV. Đệ qui

1. Khái niệm
2. Lớp các bài toán giải được bằng đệ qui
3. Các ví dụ

V. Tóm tắt nội dung bài học

VI. Bài tập

VI. Bài tập

Xem Bài 13 - Bài tập thực hành HÀM

Bài 13 - Bài thực hành: HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

I. Bài tập làm theo yêu cầu

1. Xác định vị trí tương đối của 1 điểm với tam giác ABC

Yêu cầu: Viết chương trình cho phép nhập vào tọa độ trên mặt phẳng XOY ba đỉnh của tam giác ABC, và tọa độ của điểm M; đưa ra kết luận về vị trí tương đối của M so với ABC : M nằm trong, trên cạnh hay nằm ngoài ABC.

Soạn thảo văn bản chương trình như sau

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
double KC(int x1, int y1, int x2, int y2);
double DT(double a, double b, double c);
int main()
{
    int xa,ya, xb,yb, xc,yc, xm,ym;
    double ab, bc, ac, am, bm, cm;
    double abc, amb, bmc, amc;
    printf("Nhập tọa độ điểm A(x,y):");
    scanf("%d%d", &xa, &ya);
    printf("Nhập tọa độ điểm B(x,y):");
    scanf("%d%d", &xb, &yb);
    printf("Nhập tọa độ điểm C(x,y):");
    scanf("%d%d", &xc, &yc);
    printf("Nhập tọa độ điểm M(x,y):");
    scanf("%d%d", &xm, &ym);
    ab=KC(xa,ya,xb,yb);
    bc=KC(xb,yb,xc,yc);
    ac=KC(xa,ya,xc,yc);
    am=KC(xa,ya,xm,ym);
    bm=KC(xb,yb,xm,ym);
    cm=KC(xc,yc,xm,ym);
    abc=DT(ab, bc, ac);
    amb=DT(am, bm, ab);
    bmc=DT(bm, cm, bc);
    amc=DT(am, cm, ac);
    if ((amb==0)||(bmc==0)||(amc==0))
    {
        printf("M nằm trên cạnh của ABC");
    }
    else
    {
        if ((amb+bmc+amc)==abc)
        {
```

```

        printf("M nam trong tam giac ABC");
    }
    else
    {
        printf("M nam ngoai tam giac ABC");
    }
}
getch();
return 0;
}
double KC(int x1, int y1, int x2, int y2)
{
    double kq;
    kq=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
    return kq;
}
double DT(double a, double b, double c)
{
    double kq, p;
    p=(double)(a+b+c)/2;
    kq=sqrt(p*(p-a)*(p-b)*(p-c));
    return kq;
}

```

Thử nghiệm 1: Nhấn F9 để chạy chương trình, sau đó nhập vào tọa độ cho các đỉnh của tam giác ABC lần lượt là A(x,y): 0,0 ; B(x,y): 10,0; C(x,y): 0, 10; và tọa độ của điểm M là M(x,y): 1,1. Nhận xét về kết quả đạt được.

Thử nghiệm 2: Với dữ liệu vào A(x,y): 0,0 ; B(x,y): 10,0; C(x,y): 0, 10; M(x,y): 1,1 sử dụng chức năng debug của công cụ lập trình (Dev-C++) và ghi lại giá trị của các biến abc, amb, bmc và amc. Xem lại kết luận của chương trình và đưa ra giải pháp để khắc phục.

Thử nghiệm 3: Xây dựng hàm tính bình phương của 1 số và sử dụng vào chương trình thay cho hàm sqrt() sẵn có.

2. Viết hàm đếm số từ của một chuỗi ký tự

Yêu cầu: Viết chương trình cho phép nhập vào một chuỗi ký tự, đếm số từ có trong chuỗi. Từ được hiểu là một đoạn ký tự nằm giữa hai dấu trống và khác trống.

Soạn thảo văn bản chương trình như sau

```

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
int wordCount(char *s);
int main()

```

```

{
    char s[100];
    printf("Nhap vao mot xau:");
    gets(s);
    printf("So tu: %d", wordCount(s));
    getch();
    return 0;
}
int wordCount(char *s)
{
    int kq,i;
    kq=0;
    i=0;
    while (s[i]!='\0')
    {
        if (s[i]==32)
        {
            kq=kq+1;
        }
        i++;
    }
    kq++;
    return kq;
}

```

Thử nghiệm 1: Nhấn F9 để chạy chương trình, nhập vào xâu ký tự “ky thuật lập trình” nhận xét về kết quả đạt được.

Thử nghiệm 2: Nhập vào xâu ký tự “ky thuật lập trình ” (có một ký tự trống ở cuối) nhận xét về kết quả đạt được.

Thử nghiệm 3: Nhập vào xâu ký tự “ ky thuật lập trình ” (có 1 ký tự trống ở đầu, 3 ký tự trống giữa ky và thuật, 1 ký tự trống ở cuối) nhận xét về kết quả đạt được.

II. Bài tập tự làm

- Viết 1 hàm cho phép nhập vào số nguyên, 1 hàm cho phép nhập vào số thực.
- Viết chương trình cho phép nhập vào tọa độ trên mặt phẳng XOY ba đỉnh của tam giác ABC, và tọa độ của điểm M; đưa ra kết luận về vị trí tương đối của M so với ABC : M nằm trong, trên cạnh hay nằm ngoài ABC.
- Cho tam giác ABC và một điểm M không nằm trên cạnh của tam giác. Hãy viết chương trình tính số lớn nhất R sao cho vòng tròn tâm M, bán kính R, không cắt bất cứ cạnh nào và không chứa tam giác.
- Viết hàm chuẩn hóa chuỗi họ tên người Việt theo yêu cầu sau: Không có ký tự trống phía đầu và cuối, giữa 2 từ có một ký tự trống, ký tự đầu tiên của mỗi từ được viết HOA và ký tự khác được viết thường.
- Giả sử hovaten là xâu họ tên người việt, viết các hàm TEN cho phép lấy ra phần tên, hàm HODEM cho phép lấy ra phần họ và đệm của hovaten.

6. Viết hàm đếm số từ của một chuỗi ký tự. Mỗi từ được hiểu là đoạn ký tự giữa hai dấu trống và không phải là khoảng trống.
7. Viết hàm chuyển số nguyên thành chuỗi ký tự biểu diễn cách đọc số đó. Ví dụ nhập vào số: 235 -> in ra “Hai trăm ba mươi lăm”, 807 -> in ra “Tám trăm linh bảy”, 55 -> in ra “Năm mươi lăm”.
8. Chuỗi ký tự biểu diễn ngày tháng là chuỗi dạng “DD/MM/YYYY”, trong đó DD là hai chữ số biểu diễn ngày, MM là hai chữ số biểu diễn tháng và YYYY là bốn chữ số biểu diễn năm. Viết hàm kiểm tra một chuỗi ký tự nhập vào có phải là chuỗi biểu diễn ngày tháng hay không? Nếu là chuỗi biểu diễn ngày tháng thì ngày tháng đó có hợp lệ hay không. Ví dụ chuỗi ‘30/02/2010’ là chuỗi biểu diễn ngày tháng nhưng ngày tháng đó là không hợp lệ (vì không có ngày 30 tháng 2).
9. Cho hai chuỗi biểu diễn ngày tháng sd1 và sd2, viết hàm tính số ngày tính từ sd1 đến sd2 nếu sd1 và sd2 là hai chuỗi biểu diễn ngày tháng hợp lệ (tham khảo bài 6).
10. Cho sd1 là chuỗi biểu diễn ngày tháng, n là một số nguyên. Viết hàm NgaySauN(sd1,n) cho phép trả về chuỗi ký tự dạng ngày tháng là ngày thứ n sau ngày sd1, ví dụ sd1 = “27/01/2010” và n= 15 thì hàm trả về “11/02/2010”. Viết hàm NgayTruocN(sd1,n) cho phép trả về chuỗi ký tự dạng ngày tháng là ngày thứ n trước ngày sd1, ví dụ sd1 = “27/01/2010” và n= 46 thì hàm trả về “12/12/2009”.
11. Viết hàm cho phép đổi năm dương lịch thành năm âm lịch. Ví dụ nếu năm dương lịch là 2010 thì năm âm lịch là “Canh Dần”.
12. Ta gọi tổng triệt để các chữ số của số tự nhiên n là tổng các chữ số của n nếu tổng này chỉ có một chữ số; trong trường hợp ngược lại tổng triệt để là kết quả của việc cộng dồn liên tiếp cho đến khi chỉ còn một chữ số. Ví dụ số n=1186 có tổng các chữ số là 16, số này có hai chữ số và tổng các chữ số của nó là 7. Vậy tổng triệt để của n=1186 là 7. Nhập từ bàn phím một số k, $1 \leq k \leq 9$, viết ra màn hình tất cả các số có 4 chữ số mà tổng triệt để của nó bằng k (không kể các hoán vị).
13. Một sân hình chữ nhật được chia thành $m \times n$ ô vuông có kích thước bằng nhau, với m và $n \leq 50$. Mỗi ô có thể được chôn một quả mìn. Nếu nổ một quả mìn ở ô (i,j), tức là ở hàng i và cột j, thì nó sẽ gây nổ cho các ô xung quanh (i+1,j), (i-1,j), (i,j+1), (i,j-1) nếu các ô này có mìn. Hãy nhập vào vị trí các ô có mìn và vị trí ô đầu tiên bị nổ và in ra tối đa các ô bị nổ trên sân chữ nhật.
14. Cho một dãy có n+1 ô được đánh số từ 0 đến n. Một quân cờ đứng ở ô số 0. Mỗi một nước đi quân cờ được đi lên phía trước không quá k ô. Một cách đi của quân cờ là xuất phát từ ô số 0 quân cờ phải đi đến ô thứ n. Với hai số nguyên dương n và k, hãy tính số cách đi của quân cờ. Ví dụ, với n=3 và k=2 sẽ có 3 cách đi quân cờ với các nước 1+1+1, 1+2 và 2+1. (Bài 1, Đề thi Olympic Tin học Sinh viên Đại học Tổng hợp Hà nội mở rộng, Khối không chuyên, Hà nội 4-1996).
15. Viết ra tất cả các hoán vị của dãy $X_n = \{1, 2, \dots, n\}$ có n phần tử là các số 1, 2, ..., n.
16. Tổ hợp chập k từ n, kí hiệu là C_n^k . Viết chương trình tính C_n^k biết rằng:

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k \text{ và}$$

$$C_n^0 = C_n^n = C_k^k = 1.$$

Bài 14 - CẤU TRÚC DỮ LIỆU DO NGƯỜI DÙNG TỰ ĐỊNH NGHĨA

Nội dung bài học

I. Cấu trúc dữ liệu do người dùng tự định nghĩa

1. Khái niệm
2. Khai báo biến cấu trúc
3. Các thao tác trên biến kiểu cấu trúc
4. Con trỏ cấu trúc

II. Kiểu ngăn xếp

1. Khái niệm
2. Các thao tác cơ bản
3. Cài đặt
4. Ứng dụng

III. Kiểu hàng đợi

1. Khái niệm
2. Các thao tác cơ bản
3. Cài đặt
4. Ứng dụng

IV. Tóm tắt nội dung bài học

V. Bài tập

I. Cấu trúc dữ liệu do người dùng tự định nghĩa

1 Khái niệm

Vấn đề: Làm thế nào để mô tả một SINH VIÊN với các thông tin:

- Mã số sinh viên;
- Họ tên;
- Ngày tháng năm sinh;
- Giới tính;
- Địa chỉ thường trú

Hoặc làm thế nào để mô tả NGÀY THÁNG bao gồm các thông tin:

- Ngày;
- Tháng;
- Năm

=> Hầu hết các ngôn ngữ lập trình trong đó có C/C++ cho phép người lập trình tự định nghĩa ra cấu trúc mới theo nhu cầu sử dụng từ những kiểu dữ liệu đã có hoặc đã định nghĩa trước đó.

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 sau là của kiểu cấu trúc với 7 trường

1	2	3	4	5	6	7
---	---	---	---	---	---	---

còn kiểu mảng có dạng:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----

Cú pháp 1:

```
struct <Tên cấu trúc>
{
    <Kiểu> <Tên trường 1> ;
    <Kiểu> <Tên trường 2> ;
    .....
    <Kiểu> <Tên trường n> ;
};
```

Cú pháp 2:

```
typedef struct
{
    <Kiểu> <Tên trường 1> ;
    <Kiểu> <Tên trường 2> ;
    .....
    <Kiểu> <Tên 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).

Ví 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 KieuNgayThang
{
    unsigned char Ngay;
    unsigned char Thang;
    unsigned int Nam;
};
typedef struct
{
    unsigned char Ngay;
    unsigned char Thang;
    unsigned int Nam;
} KieuNgayThang;
```


Ví 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 KieuSinhVien
{
char MSSV[10];
char HoTen[40];
struct KieuNgayThang NgaySinh;
int Phai;
char DiaChi[40];
};
```

```
typedef struct
{
char MSSV[10];
char HoTen[40];
KieuNgayThang NgaySinh;
int Phai;
char DiaChi[40];
} KieuSinhVien;
```

- Mỗi thành phần giống là một biến riêng thuộc cấu trúc, nó gồm kiểu và tên thành phần. Một thành phần cũng còn được gọi là trường.
- Phần tên của kiểu cấu trúc và phần danh sách biến có thể có hoặc không. Tuy nhiên trong khai báo kí tự kết thúc cuối cùng phải là dấu chấm phẩy (;).
- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có thể lại là một trường có kiểu cấu trúc.
- Một biến có kiểu cấu trúc sẽ được cấp phát bộ nhớ sao cho các thực hiện của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.

2. 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>...];
```

Ví dụ: Khai báo biến NgaySinh có kiểu cấu trúc KieuNgayThang; biến SV có kiểu cấu trúc KieuSinhVien.

```
struct KieuNgayThang NgaySinh;
struct KieuSinhVien SV;
KieuNgayThang NgaySinh;
KieuSinhVien SV;
```

3. Các thao tác trên biến kiểu cấu trúc

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>.

Ví dụ 1: Viết chương trình cho phép đọc dữ liệu từ bàn phím cho biến mẫu tin SinhVien và in biến mẫu tin đó lên màn hình:

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
typedef struct
{
    unsigned char Ngay;
    unsigned char Thang;
    unsigned int Nam;
} KieuNgayThang;
typedef struct
{
    char MSSV[10];
    char HoTen[40];
    KieuNgayThang NgaySinh;
    int Phai;
    char DiaChi[40];
} KieuSinhVien;
/* Hàm in lên màn hình 1 m?u tin SinhVien*/

void InSV(KieuSinhVien 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);
}
int main()
{
    KieuSinhVien 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;
    InSV(s);
    getch();
    return 0;
}
```

}

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.

Ví 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 $c_1 = (a_1, b_1)$ và $c_2 = (a_2, b_2)$ khi đó tổng của hai số phức c_1 và c_2 là một số phức c_3 mà $c_3 = (a_1 + a_2, b_1 + b_2)$. 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;
/* Ham in so phuc len man hinh*/
void InSoPhuc(SoPhuc p)
{
    printf("%.2f + i%.2f\n",p.Thuc,p.Ao);
}
int 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();
return 0;
}

```

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 (,).

Ví dụ: Khởi tạo biến cấu trúc NgaySinh:

```
struct KieuNgayThang NgaySinh = {29, 8, 1986};
```

4. Con trỏ cấu trúc

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>;
```

Ví dụ: Ta có thể khai báo một con trỏ cấu trúc kiểu NgayThang như sau:

```
struct NgayThang *p;
/* NgayThang *p; // Nếu có định nghĩa kiểu */
```

Sử dụng các con trỏ kiểu cấu trúc

Khi khai báo biến con trỏ cấu trúc, biến con trỏ chưa có 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 cụ thể. 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.

Ví 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.

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.

Ví 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;
int 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();
    return 0;
}

```

- Cho phép sử dụng cấu trúc, con trỏ cấu trúc là đối số của hàm như các loại biến khác
- Cho phép hàm xây dựng trả về là kiểu cấu trúc

Ví dụ : Xử lý danh sách sinh viên, sử dụng hàm với đối số là cấu trúc

```

#include <stdio.h>
#include <conio.h>
//Ngay thang
struct Ngaythang {
    int ng ;
    int th ;
    int nam ;
};
//Sinh vien
struct Sinhvien {
    char hoten[25] ;
    Ngaythang ns;
    int gt;
    float diem ;
};
//cac ham xu ly
int sua(Sinhvien &r) ;
int in(Sinhvien x) ;
int nhap(Sinhvien *p) ;
int nhapds(Sinhvien *a, int n) ;
int suads(Sinhvien *a, int n) ;
int inds(const Sinhvien *a, int n) ;
//
struct Sinhvien a[10];

int main()
{
    nhap(a);
}

```

```

// in(a[1]);
// sua(a[1]);
  nhapds(a,9);
  suads(a,9);
  inds(a,9);
  getch();
  return 0;
}
///trien khai cac ham
int sua(Sinhvien &r)
{
  int chon;
  do {
    printf("1: Sua ho ten\n2: Sua ngay sinh\n3:Sua gioi tinh\n4:Sua diem\n5:ln\n0: Ket thuc\n");
    scanf("%d",&chon);
    switch (chon)
    {
      case 1:
        printf("Nhap ho ten:");
        scanf("%s",r.hoten);
        break;
      case 2:
        printf("Nhap ngay thang nam sinh:");
        scanf("%d%d%d",&r.ns.ng,&r.ns.th,&r.ns.nam);
        break;
      case 3:
        printf("Nhap gioi tinh 0:Nu 1:Nam:");
        scanf("%d",&r.gt) ;
        break;
      case 4:
        printf("Nhap diem:");
        scanf("%f",&r.diem);
        break;
      case 5:
        printf("Sinh vien:");
        in(r);
        break;
      case 0:
        break;
      default:
        printf("Nhap gia tri khong dung\n");
        break;
    }
  } while (chon) ;
  return 0;
}
/////
int in(Sinhvien x)
{
  printf("Ho ten :%s\nNgay sinh %d/%d/%d\n",x.hoten,x.ns.ng, x.ns.th, x.ns.nam) ;
  printf("Gioi tinh :%s\nDiem :%f\n",(x.gt==1) ?"Nam" : "Nu",x.diem) ;
  return 0;
}

```

```

/////
int nhap(Sinhvien *p)
{
    printf("Nhap ho va ten: ");
    scanf("%s",p->hoten);
    printf("Nhap ngay sinh (ngay thang nam): ");
    scanf("%d%d%d", &p->ns.ng ,&p->ns.th,&p->ns.nam);
    printf("Gioi tinh 0: nu, 1: nam: ");
    scanf("%d",& p->gt);
    printf("Diem: ");
    scanf("%f",& p->diem);
    return 0;
}
/////
int nhapds(Sinhvien *a, int n)
{
    for (int i=1; i<=n; i++) nhap(&a[i] );
    return 0;
}
/////
int suads(Sinhvien *a, int n)
{
    int chon;
    do
    {
        printf("\nNhap phan tu duoc su tu 1 den %d, gia tri khac thoat:",n);
        scanf("%d",&chon);
        if(chon>0 &&chon<=n)
        {
            sua(a[chon] );
        }
    }while(chon>0 && chon<=n);
    return 0;
}
//////////
int inds(const Sinhvien *a, int n)
{
    for (int i=1; i<=n; i++)
        in(a[i] );
    return 0;
}

```

5. Cấu trúc với thành phần kiểu bit

a. Trường bit

Để tiết kiệm trong lưu trữ, trong ngôn ngữ lập trình C cho phép khai báo các trường của cấu trúc với số lượng bit xác định không phụ thuộc vào số lượng bit các kiểu dữ liệu chuẩn.

Một trường bit là một khai báo trường int và thêm dấu: cùng số bit n theo sau, trong đó $0 \leq n < 15$. Ví dụ do độ lớn của ngày không vượt quá 31, tháng không vượt quá 12 nên

2 trường này trong cấu trúc ngày tháng có thể khai báo tiết kiệm hơn bằng 5 và 4 bit như sau:

```
struct Date {
    int ng: 5;
    int th: 4;
    int nam:14;
};
```

b. Đặc điểm

Cần chú ý các đặc điểm sau của một cấu trúc có chứa trường bit:

- Các bit được bố trí liên tục trên dãy các byte.
- Kiểu trường bit phải là int (signed hoặc unsigned).
- Độ dài mỗi trường bit không quá 16 bit.
- Có thể bỏ qua một số bit nếu bỏ trống tên trường, ví dụ:

```
struct tu {
    int: 8;
    int x:8;
}
```

mỗi một biến cấu trúc theo khai báo trên gồm 2 byte, bỏ qua không sử dụng byte thấp và trường x chiếm byte (8 bit) cao.

- Không cho phép lấy địa chỉ của thành phần kiểu bit.
- Không thể xây dựng được mảng kiểu bit.
- Không được trả về từ hàm một thành phần kiểu bit. Ví dụ nếu b là một thành phần của biến cấu trúc x có kiểu bit thì câu lệnh sau là sai:

```
return x.b ; // sai
```

tuy nhiên có thể thông qua biến phụ như sau:

```
int tam = x.b ; return tam ;
```

- Tiết kiệm bộ nhớ
- Dùng trong kiểu union để lấy các bit của một từ (xem ví dụ trong phần kiểu hợp).

6. Câu lệnh typedef

Để thuận tiện trong sử dụng, thông thường các kiểu được NSD tạo mới sẽ được gán cho một tên kiểu bằng câu lệnh typedef như sau:

```
typedef <kiểu> <tên_kiểu> ;
```

Ví dụ: Để tạo kiểu mới có tên Bool và chỉ chứa giá trị nguyên (thực chất chỉ cần 2 giá trị 0, 1), ta có thể khai báo:

```
typedef int Bool;
```

khai báo này cho phép xem Bool như kiểu số nguyên.

hoặc có thể đặt tên cho kiểu ngày tháng là Date với khai báo sau:

```
typedef struct Date {
    int ng;
```



```
int th;
int nam;
};
```

khi đó ta có thể sử dụng các tên kiểu này trong các khai báo (ví dụ tên kiểu của đối, của giá trị hàm trả lại ...).

7. Hàm sizeof()

Hàm trả lại kích thước của một biến hoặc kiểu. Ví dụ:

```
Bool a, b;
Date x, y, z[50];
printf("%d,%d,%d",sizeof(a),sizeof(b),sizeof(Bool)); // in 2,2,2
printf("Số phần tử của z =%d ",sizeof(z) / sizeof(Date)); // in 50
```

II. Ngăn xếp

- Khai báo biến cố định làm lãng phí bộ nhớ do khai báo số lượng phần tử lớn nhất để lưu trữ dữ liệu

- Khi thay đổi về số lượng phần tử cần phải lưu trữ thì mảng cấp phát động không thể giải quyết được tiết kiệm bộ nhớ và hạn chế các thao tác chuyển bộ nhớ.

- Giải quyết vấn đề cấp phát bộ nhớ của khối dữ liệu liên tục sẽ vượt quá giới hạn quản lý bộ nhớ của một số hệ điều hành, khó trong tìm được vùng nhớ lớn phù hợp để cấp phát.

- Giải pháp đưa ra kiểu cấu trúc, bản thân các đối tượng trữ (liên kết) đến đối tượng tiếp theo trong danh sách.

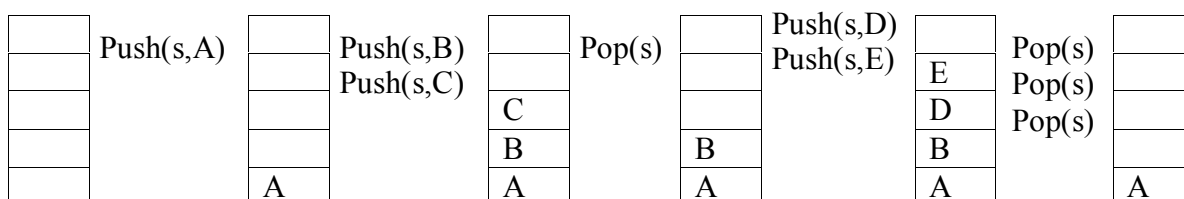
- Các vùng nhớ cấp phát không liên tục nên các lệnh làm việc với nhóm các đối tượng sẽ không thực hiện được.

Kiểu cấu trúc liên kết được sử dụng xây dựng nhiều dạng cấu trúc khác nhau, nhưng đặc trưng là cấu trúc ngăn xếp và hàng đợi.

1. Khái niệm

Ngăn xếp là kiểu đặc biệt của danh sách việc loại bỏ và bổ sung đều được thực hiện trên một đầu của danh sách. Vì thế các phần tử khi bổ sung vào ngăn xếp sau sẽ được lấy ra trước nên gọi là LIFO (Last In First Out). Thao tác cơ bản của ngăn xếp là bổ sung (push), loại bỏ (pop), ngoài ra có thao tác khởi tạo, kiểm tra trạng thái.

Ví dụ trực quan của ngăn xếp là chồng bát, bát mới vào luôn được đặt lên trên, thứ tự lấy bát từ trên xuống dưới.



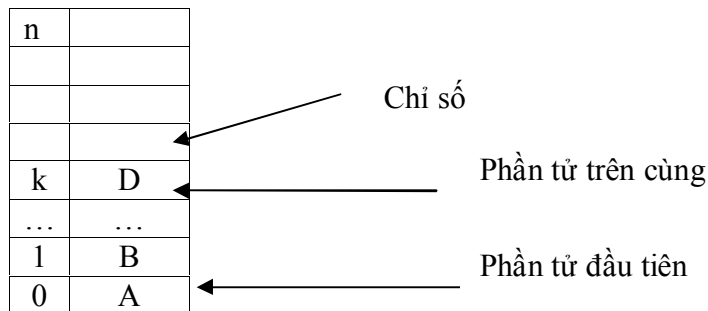
- Các thao tác cơ bản với ngăn xếp

- Khởi tạo
- Kiểm tra ngăn xếp rỗng

- Bổ sung một phần tử vào ngăn xếp
- Lấy một phần tử khỏi ngăn xếp

2. Cài đặt ngăn xếp sử dụng mảng

Sử dụng mảng một chiều để thể hiện ngăn xếp, dùng một biến chỉ số mô tả phần tử ở đỉnh của ngăn xếp. Biến chỉ số sẽ chỉ ra vị trí tiếp theo được thêm vào, và vị trí trừ đi 1 là vị trí phần tử đang có ở ngăn xếp.



Khai báo

```
typedef struct stack
{
    int top;
    int storage[MAX];
};
```

Khởi tạo ngăn xếp

```
void init(stack *s)
{
    s->top=0;
}
```

Kiểm tra ngăn xếp rỗng

```
///0: empty
int empty(stack *s)
{
    return s->top;
}
```

Kiểm tra ngăn xếp đầy

```
///0:full
int full(stack *s)
{
    if(s->top>=MAX)
    {
        return 0;
    }
    else
    {
        return s->top+1;
    }
}
```

Bổ sung một phần tử vào ngăn xếp

```
//-1 full of stack
int push(stack *s, int value)
{
    if(s->top==MAX)
```

```

    {
        return -1;
    }
    else
    {
        s->storage[s->top]=value;
        s->top++;
    }
    return s->top;
}

```

Lấy một phần tử khỏi ngăn xếp

```

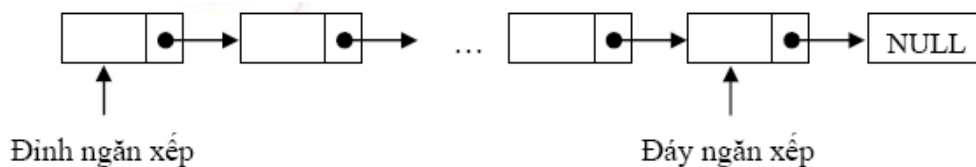
// -1: try to pop the empty stack
int pop(stack *s, int *out)
{
    if(s->top<=0)
    {
        out=0;
        return -1;
    }
    else
    {
        s->top--;
        *out=s->storage[s->top];
        return s->top;
    }
}

```

Do ngăn xếp được triển khai dạng mảng nên gặp phải vấn đề phải khai báo khối dữ liệu tối đa để sử dụng, trong trường hợp phát triển nếu số lượng vượt quá số lượng MAX (vẫn trong giới hạn cho phép của bộ nhớ) thì chương trình sẽ không thực hiện được.

3. Cài đặt ngăn xếp sử dụng con trỏ liên kết

Cấp phát đúng số lượng phần tử mà ngăn xếp đang cần. Thao tác phức tạp.



Khai báo

```

typedef struct stackNode
{
    int item;
    stackNode *next;
};
typedef struct stack{
    stackNode* top;
};

```

Khởi tạo ngăn xếp

```

void init(stack *s)
{
    s->top=NULL;
}

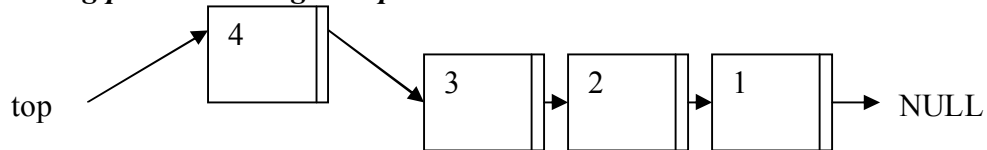
```

```

}
Kiểm tra ngăn xếp rỗng
///0: empty
int empty(stack *s)
{
    if(s->top==NULL)
    {
        return 0;
    }
    else
    {
        return 1;
    }
}
}

```

Bổ sung phần tử vào ngăn xếp

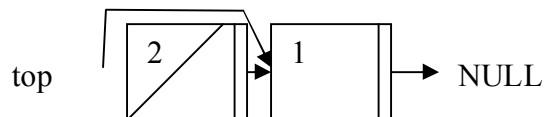


```

///-1: cant push to stack
int push(stack *s,int value)
{
    stackNode *a;
    a=(stackNode*)malloc(sizeof(stackNode));
    if(a==NULL)
    {
        return -1;
    }
    a->item=value;
    a->next=s->top;
    s->top=a;
    return 0;
}

```

Lấy một phần tử ra khỏi ngăn xếp



```

int pop(stack *s,int *out)
{
    stackNode *cur;
    if(s->top==NULL)
    {
        return -1;
    }
    *out=s->top->item;
    cur=s->top;
    s->top=s->top->next;
}

```

```
    free(cur);  
}
```

4. Một số ứng dụng của ngăn xếp

Ngăn xếp được sử dụng trong nhiều ứng dụng khác nhau

- Đảo ngược xây ký tự
- Tính giá trị biểu thức dạng hậu tố
- Chuyển biểu thức dạng trung tố sang hậu tố
- Sử dụng để khử đệ qui, chuyển bài toán đệ qui (sử dụng ngăn xếp của hệ thống) sang sử dụng ngăn xếp ngoài.

Đảo ngược xây ký tự

Ví dụ về đảo ngược xây sử dụng ngăn xếp. Do ngăn xếp có qui luật vào trước ra sau nên lần lượt đưa các ký tự vào ngăn xếp, sau đó tiến hành lấy ra các ký tự sẽ được đảo ngược thứ tự.

```
#include <stdio.h>  
#include <conio.h>  
#include <stdlib.h>  
#include <string.h>  
typedef struct stackNode  
{  
    char item;  
    stackNode *next;  
};  
typedef struct stack{  
    stackNode* top;  
};  
///  
void init(stack *s);  
///  
//0: empty  
int empty(stack *s);  
///  
//-1: cant push to stack  
int push(stack *s,char value);  
///  
//-1: pop empty stack  
int pop(stack *s,char *out);
```

```
stack s;  
int main()  
{  
    int i;  
    char v;  
    char *c="xau dao nguoc";  
    init(&s);  
    for(i=0;i<strlen(c);i++)  
    {  
        push(&s,c[i]);  
    }  
    while(empty(&s)!=0)  
    {  
        pop(&s,&v);  
        printf("\n%c",v);  
    }  
}
```

```

    getch();
    return 0;
}
//implementation
void init(stack *s)
{
    s->top=NULL;
}
//-----
//0: empty
int empty(stack *s)
{
    if(s->top==NULL)
    {
        return 0;
    }
    else
    {
        return 1;
    }
}
//-----
// -1: cant push to stack
int push(stack *s,char value)
{
    stackNode *a;
    a=(stackNode*)malloc(sizeof(stackNode));
    if(a==NULL)
    {
        return -1;
    }
    a->item=value;
    a->next=s->top;
    s->top=a;
    return 0;
}
//-----
// -1: pop empty stack
int pop(stack *s,char *out)
{
    stackNode *cur;
    if(s->top==NULL)
    {
        return -1;
    }
    *out=s->top->item;
    cur=s->top;
    s->top=s->top->next;
    free(cur);
}

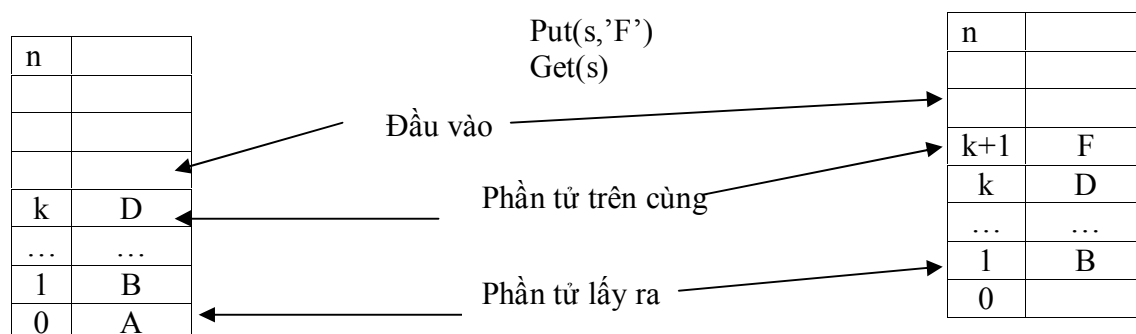
```

III. Hàng đợi

1. Khái niệm

Hàng đợi là cấu trúc dữ liệu tương tự ngăn xếp nhưng nguyên tắc đưa vào và lấy ra các phần tử theo thứ tự phần tử đưa vào trước sẽ lấy ra trước FIFO (First In First Out). Qui luật này thể hiện các mô hình trong thực tế như hàng đợi mua vé, hàng hóa trong kho, ...

Do các phần tử vào trước được lấy ra trước nên cần phải xây dựng danh sách có hai đầu, đầu vào và đầu ra.

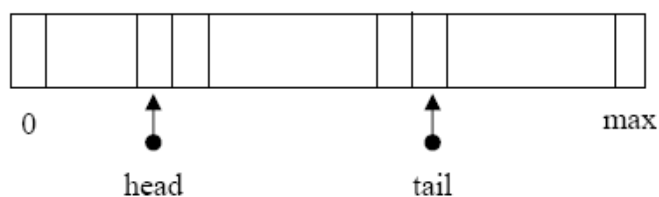


– Các thao tác cơ bản với hàng đợi

- Khởi tạo hàng đợi
- Kiểm tra hàng đợi rỗng
- Bổ sung một phần tử vào hàng đợi
- Lấy một phần tử khỏi hàng đợi

2. Cài đặt hàng đợi sử dụng mảng

Có thể sử dụng mảng để lưu trữ dữ liệu, vì thế có thể sử dụng để mô tả hàng đợi. Do hàng đợi có hai đầu nên cần phải có hai chỉ số để đánh dấu. Do việc lấy dữ liệu và đưa dữ liệu vào trong danh sách đều tăng chỉ số nên cần phải kiểm tra trong trường hợp chỉ số vượt qua giới hạn của mảng cần phải chuyển chỉ số về 0.



Khởi tạo

```
typedef struct queue
{
    int head, tail;
    int storage[MAX];
};
```

Khởi tạo hàng đợi

```
void init(queue *s)
{
    s->tail=0;
    s->head=0;
}
```

Kiểm tra hàng đợi rỗng

```
///0: empty
int empty(queue *s)
{
    int d=s->head-s->tail;
    if(d<0)
        d+=MAX;
    return d;
}
```

Kiểm tra hàng đợi đầy

```
///0:full
int full(queue *s)
{
    int nh=s->head+1;
    if(nh==s->tail || (nh + MAX)%MAX==s->tail)
    {
        return 0;
    }
    return 1;
}
```

Thêm một phần tử vào hàng đợi

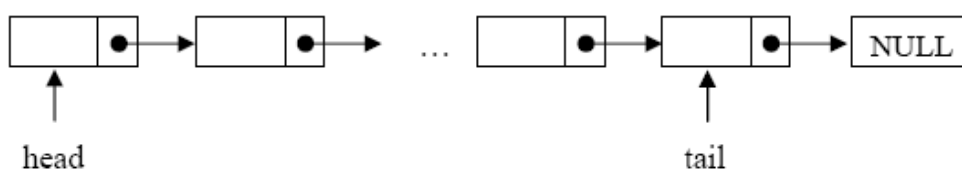
```
//-1 full of queue
int put(queue *s, int value)
{
    if(full(s)==0)
        return -1;
    s->storage[s->head]=value;
    s->head++;
    s->head=s->head%MAX;
    return s->head;
}
```

Lấy một phần tử khỏi hàng đợi

```
//-1: try to pop the empty queue
int get(queue *s, int *out)
{
    if(empty(s)==0)
        return -1;
    *out=s->storage[s->tail];
    s->tail++;
    s->tail=s->tail%MAX;
    return s->tail;
}
```

3. Cài hàng đợi sử dụng con trỏ

Sử dụng hai con trỏ là đầu và đuôi. Khi thêm nối vào sau con trỏ đuôi, và khi lấy ra sẽ từ con trỏ đầu.



Khai báo

```
typedef struct queueNode
{
    int item;
    queueNode *next;
};
typedef struct queue{
    queueNode* head, *tail;
};
```

Khởi tạo hàng đợi

```
void init(queue *s)
{
    s->head=NULL;
    s->tail=NULL;
}
```

Kiểm tra hàng đợi rỗng

```
///0: empty
int empty(queue *s)
{
    if(s->head==NULL)
    {
        return 0;
    }
    else
    {
        return 1;
    }
}
```

Thêm một phần tử vào hàng đợi

```
///-1: cant push to queue
int put(queue *s,int value)
{
    queueNode *a;
    a=(queueNode*)malloc(sizeof(queueNode));
    if(a==NULL)
    {
        return -1;
    }
    a->item=value;
    a->next=NULL;
    if(s->tail==NULL)
    {
        s->tail=a;
        s->head=a;
    }
    s->tail->next=a;
    s->tail=a;
    return 0;
}
```

Lấy một phần tử khỏi hàng đợi

```
///-1: pop empty queue
int get(queue *s,int *out)
{

```

```

queueNode *a;
if(s->head==NULL)
{
    return -1;
}
*out=s->head->item;
a=s->head;
s->head=s->head->next;
if(s->head==NULL)
{
    s->tail=NULL;
}
free(a);
}

```

4. Một số ứng dụng của hàng đợi

- Hàng đợi được sử dụng trong mô phỏng các mô hình hàng đợi thực tế trong các bài toán
- Sử dụng trong bài toán duyệt theo chiều rộng

IV. Tóm tắt nội dung bài học

I. Cấu trúc tự định nghĩa

1. Khái niệm
2. Khai báo biến cấu trúc
3. Các thao tác trên biến kiểu cấu trúc
4. Con trỏ cấu trúc

II. Kiểu ngăn xếp

1. Khái niệm
2. Các thao tác cơ bản
3. Cài đặt
4. Ứng dụng

III. Kiểu hàng đợi

1. Khái niệm
2. Các thao tác cơ bản
3. Cài đặt
4. Ứng dụng

V. Bài tập

Xem Bài 15 - Bài tập thực hành Cấu trúc dữ liệu tự định nghĩa.

Bài 15 - Bài thực hành: CẤU TRÚC DỮ LIỆU DO NGƯỜI DÙNG TỰ ĐỊNH NGHĨA

I. Bài tập làm theo yêu cầu

1. Biểu diễn và thực hiện các phép toán phân số

Yêu cầu: Xây dựng cấu trúc biểu diễn phân số; cho phép nhập vào 2 phân số, thực hiện các phép cộng, trừ, nhân, chia 2 phân số đó.

Soạn thảo văn bản chương trình như sau

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

typedef struct fraction
{
    int numerator;
    int denominator;
};

//-- khai bao cac ham
fraction add(fraction x, fraction y) ;
fraction sub(fraction x, fraction y) ;
fraction multiply(fraction x, fraction y) ;
fraction divide(fraction x, fraction y) ;
void printFraction(fraction x);

int main()
{
    fraction a,b;

    printf("Nhap phan so, tu so truoc, mau so sau:");
    scanf("%d%d",&a.numerator, &a.denominator);
    printf("Nhap phan so, tu so truoc, mau so sau:");
    scanf("%d%d",&b.numerator, &b.denominator);
    printf("Hieu:\n");
    printFraction(sub(a,b));
    printf("\nTong:\n");
    printFraction(add(a,b));
    printf("\nTich:\n");
    printFraction(multiply(a,b));
    printf("\nThuong:\n");
    printFraction(divide(a,b));
    getch();
    return 0;
}

//----- trien khai cac ham
fraction add(fraction x, fraction y)
{
    fraction t;
    t.numerator=x.numerator * y.denominator + x.denominator * y.numerator;
```

```

        t.denominator = x.denominator * y.denominator;
        return t;
    }
    ///-----
    fraction sub(fraction x, fraction y)
    {
        fraction t;
        t.numerator=x.numerator * y.denominator - y.numerator * x.denominator;
        t.denominator = x.denominator * y.denominator;
        return t;
    }
    fraction multiply(fraction x, fraction y)
    {
        fraction t;
        t.numerator=x.numerator * y.numerator;
        t.denominator = x.denominator * y.denominator;
        return t;
    }
    fraction divide(fraction x, fraction y)
    {
        fraction t;
        t.numerator=x.numerator * y.denominator;
        t.denominator = x.denominator * y.numerator;
        return t;
    }
    void printFraction(fraction x)
    {
        printf("%d/%d",x.numerator,x.denominator);
    }
}

```

Thử nghiệm 1:

```

Nhap phan so, tu so truoc, mau so sau:3 5
Nhap phan so, tu so truoc, mau so sau:7 11
Hieu:
-2/55
Tong:
68/55
Tich:
21/55
Thuong:
33/35

```

Thử nghiệm 2:

Thử nghiệm với hai giá trị là $3/4$, và $1/2$ đưa ra nhận xét, cần cải tiến như thế nào?

Thử nghiệm 3:

Thử nghiệm với hai giá trị là $3/4$, và $0/2$ đưa ra nhận xét, cần cải tiến như thế nào?

2. Chuyển biểu thức trung tố về dạng hậu tố

Yêu cầu: Viết chương trình cho phép chuyển một chuỗi biểu diễn biểu thức dạng trung tố thành chuỗi biểu diễn biểu thức dạng hậu tố.

Soạn thảo văn bản chương trình như sau

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
typedef struct stackNode
{
    char item;
    stackNode *next;
};
typedef struct stack{
    stackNode* top;
};
///
void init(stack *s);
///0: empty
int empty(stack *s);
///-1: cant push to stack
int push(stack *s,char value);
///-1: pop empty stack
int pop(stack *s,char *out);
char getTop(stack *s);
int isnumber(char ch);
int isoperator(char ch);
int priority(char ch);
int convert(char source[], char chuoihauto[]);

stack s;
int main()
{
    int i;
    char v;
    char c[100];
    char o[100];

    gets(c);
    convert(c,o);
    puts(o);
    getch();
    return 0;
}
///implementation
void init(stack *s)
{
    s->top=NULL;
}
///-----
```

```

///0: empty
int empty(stack *s)
{
    if(s->top==NULL)
    {
        return 0;
    }
    else
    {
        return 1;
    }
}
///-----
///-1: cant push to stack
int push(stack *s,char value)
{
    stackNode *a;
    a=(stackNode*)malloc(sizeof(stackNode));
    if(a==NULL)
    {
        return -1;
    }
    a->item=value;
    a->next=s->top;
    s->top=a;
    return 0;
}
///-----
///-1: pop empty stack
int pop(stack *s,char *out)
{
    stackNode *cur;
    if(s->top==NULL)
    {
        return -1;
    }
    cur=s->top;
    *out=s->top->item;
    s->top=cur->next;
    free(cur);
}
///-----
char getTop(stack *s)
{
    return s->top->item;
}
///-----
int isnumber(char ch)
{
    return ((int(ch)>=48&&int(ch)<=57));
}
///-----
int isoperator(char ch)

```

```

{
    return (ch=='+'||ch=='-'||ch=='*'||ch=='/');
}
///-----
int priority(char ch)
{
    if(ch=='+'||ch=='-')
        return 0;
    if(ch=='*'||ch=='/')
        return 1;
}
///-----
int convert(char source[], char suffixExp[])
{
    stack S;
    init(&S);
    int count=0;
    char temp;
    for(int i=0;i<strlen(source);i++)
    {
        if(isoperator(source[i])&&isoperator(source[i+1]))
            return i;
        if((source[i]>=65&&source[i]<=90)||((source[i]>=97&&source[i]<=121)))
            return i;
    }
    for(int i=0;i<strlen(source);i++)
    {
        char ch=source[i];
        if(isnumber(ch))
        {
            suffixExp[count]=ch;
            count++;
        }
        else//operator
        {
            if(empty(&S)!=0)//not empty
            {
                if(isoperator(getTop(&S))&&(priority(getTop(&S))>=priority(ch)))
                {
                    pop(&S,&temp);
                    suffixExp[count++]=temp;
                    push(&S,ch);
                }
                else
                    push(&S,ch);
            }
            else
                push(&S,ch);
        }
    }

}

} //END FOR
while(empty(&S)!=0)//Lay het nhung gi con lai trong stack

```

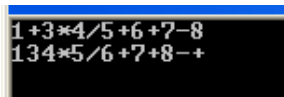
```

    {
        pop(&S,&temp);
        suffixExp[count++]=temp;
    }
    suffixExp[count++]='\0';//CHEN KI TU NULL VAO CUOI XAU;
}

```

Thử nghiệm 1:

Nhập chuỗi là: 1-3*4/5+6+7-8



Thử nghiệm 2:

Nhập chuỗi là: 1+3-2*3

Nhận xét và phát triển cho trường hợp có dấu ngoặc (,), phép toán ^

II. Bài tập tự làm

1. Hãy định nghĩa kiểu:

```

struct Hosoi{
    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, 10 Giỏi

7, 8 Khá

5, 6 Trung bình

dưới 5 Không đạt

In danh sách lên màn hình theo dạng sau:

XEP LOAI VAN HOA

HO VA TEN	DIEM	XEPLOAI
Nguyen Van A	7	Kha
Ho Thi B	5	Trung binh
Dang Kim C	4	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, Dchi. 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	333.00	26	Can Tho
2	Dang Kim B	290.00	23	Vinh Long

4. Hoàn thiện mã nguồn của chương trình tính biểu thức dạng hậu tố.
5. Hoàn thiện mã nguồn chương trình chuyển đổi biểu thức dạng trung tố sang hậu tố
6. Viết chương trình đổi số nguyên dạng thập phân sang dạng nhị phân sử dụng ngăn xếp
7. Cho một ma trận mìn $M \times N$, các điểm nhận giá trị 0 và 1, 1 là ô có mìn. Nếu nỏ ở ô (i,j) sẽ kích nỏ bốn ô xung quanh nếu ô đó có mìn. Cho nỏ ở ô (i,j) hãy chỉ ra các ô bị kích nỏ khác. Sử dụng ngăn xếp để xét các ô bị kích nỏ tiếp theo, in ra thứ tự các ô. Sử dụng hàng đợi để xét các ô bị kích nỏ tiếp theo, in ra thứ tự các ô.

Bài 16 - LÀM VIỆC VỚI FILE

Nội dung bài học

I. Một số khái niệm

II. Các thao tác trên tập tin

1. Khai báo biến tập tin

2. Mở tập tin

3. Đóng tập tin

4. Kiểm tra đến cuối tập tin hay chưa?

5. Di chuyển con trỏ tập tin về đầu tập tin - Hàm `rewind()`

III. Truy cập tập tin văn bản

1. Ghi dữ liệu lên tập tin văn bản

2. Đọc dữ liệu từ tập tin văn bản

IV. Truy cập tập tin nhị phân

1. Ghi dữ liệu lên tập tin nhị phân - Hàm `fwrite()`

2. Đọc dữ liệu từ tập tin nhị phân - Hàm `fread()`

3. Di chuyển con trỏ tập tin - Hàm `fseek()`

4. Ví dụ

V. Tóm tắt nội dung bài học

I. Một số khái niệm

Dữ liệu trong chương trình được lưu trữ ở RAM máy tính, vì thế khi kết thúc chương trình, tất cả dữ liệu sẽ bị giải phóng. Để xử lý dữ liệu cần phải lưu trữ trên bộ nhớ ngoài (đĩa cứng, USB, ...) dưới dạng file. File là tập hợp các byte liên tục được lưu trữ và được gán tên gọi. Khi xử lý file chương trình có thể xem xét chuỗi byte với cách nhìn khác nhau, có những ứng xử khác nhau với dữ liệu. Trong C hỗ trợ việc thao tác với file với quan điểm: file văn bản, file nhị phân.

- File 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).

Truy xuất tập tin theo kiểu văn bản chỉ có thể truy xuất theo kiểu tuần tự.

- Tập tin nhị phân: Quan điểm tập tin là dãy byte liên tục, việc xử lý với tập tin dựa trên việc đọc ghi dãy byte.

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

Các thao tác với tập tin:

- 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().

Ở đây, ta thao tác với tập tin nhờ các hàm được định nghĩa trong thư viện stdio.h.

1. Khai báo biến tập tin

Cú pháp

FILE <Danh sách các biến con trỏ, đại diện cho tập tin>

Các biến trong danh sách phải là các con trỏ và được phân cách bởi dấu phẩy(.).

Ví dụ: FILE *f1,*f2;

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.

- Mode: 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

Mặc định là mở dạng text nếu không có xác định là b, nếu rõ ràng hơn thì thêm chỉ định t để xác định là kiểu text.

- 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.

Ví 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*/
}
```

Kiểm tra con trỏ f với giá trị NULL cho phép xác định được lệnh thực hiện thành công hay không?

Nếu mở tập tin để ghi, trường hợp 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.

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.

Ngoài ra, ta còn có thể sử dụng hàm fcloseall() để đóng tất cả các tập tin lại.

Cú pháp: int fcloseall()

Kết quả trả về của hàm là tổng số các tập tin được đóng lại. Nếu không thành công, kết quả trả về là EOF.

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.

5. Di chuyển con trỏ tập tin về đầu tập tin - Hàm rewind()

- Chuyển về đầu tập tin, sử dụng hàm rewind().

Cú pháp: *void rewind(FILE *f);*

+ f: con trỏ tập tin đang thao tác.

- Chuyển đến vị trí bất kỳ sử dụng hàm `fseek()`.

Cú pháp: *int fseek(FILE *stream, long offset, int whence);*

+ stream: con trỏ tập tin đang thao tác.

+ offset: số byte cần dịch chuyển con trỏ tập tin.

+ whence: vị trí bắt đầu để tính khoảng cách dịch chuyển cho offset:

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.

- Lấy vị trí của con trỏ file hàm `ftell()`;

Cú pháp: *long ftell(FILE *stream);*

+ stream: biến đại diện cho file

+ trả về vị trí của con trỏ file so với đầu file

III. Truy cập tập tin văn bản

1. Ghi dữ liệu lên tập tin văn bản

1.1 Hàm `fputc()`

Hàm này được dùng để ghi một ký tự lên một tập tin văn bản đang được mở để làm việc.

Cú pháp: *int fputc(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.

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 fputs(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.

1.3 Hàm `fprintf()`

Hàm này dùng để ghi dữ liệu có định dạng lên tập tin văn bản.

Cú pháp: *fprintf(FILE *f, const char *format, vaxpr)*

Trong đó: format: chuỗi định dạng (giống với các định dạng của hàm `printf()`), vaxpr: danh sách các biểu thức, mỗi biểu thức cách nhau dấu phẩy (,).

Định dạng	Ý nghĩa
-----------	---------

%d	Ghi số nguyên
%[.số chữ số thập phân] f	Ghi số thực có <số chữ số thập phân> theo quy tắc làm tròn số.
%o	Ghi số nguyên hệ bát phân
%x	Ghi số nguyên hệ thập lục phân
%c	Ghi một ký tự
%s	Ghi chuỗi ký tự
%e hoặc %E hoặc %g hoặc %G	Ghi số thực dạng khoa học (nhân 10 mũ x)

Ví dụ: Viết chương trình ghi chuỗi ký tự lên tập tin văn bản D:\\Baihat.txt

```
#include<stdio.h>
#include<conio.h>
int main(){
FILE *f;
clrscr();
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();
return 0;
}
```

2. Đọc dữ liệu từ tập tin văn bản

2.1 Hàm fgetc()

Hàm này dùng để đọc dữ liệu từ tập tin văn bản đang được mở để làm việc.

Cú pháp: *int fgetc(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.

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 đệm.
- 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.

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: *fscanf(FILE *f, const char *format, varlist)*

Trong đó: format: chuỗi định dạng (giống hàm scanf()); varlist: danh sách các biến mỗi biến cách nhau dấu phẩy (,).

Ví dụ: Viết chương trình chép tập tin D:\Baihat.txt ở trên sang tập tin D:\Baica.txt.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *f1, *f2;
    clrscr();
    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);
        }
        fcloseall();
    }
    getch();
    return 0;
}
```

3. Ví dụ

Đọc ma trận tính ma trận, ghi ma trận chuyển vị vào file mới. Xem xét file văn bản như đầu vào, đầu ra của chương trình, thay vì nhận từ bàn phím và xuất ra màn hình.

```
#include<stdio.h>
#include<conio.h>
int a[100][100];
int n,m;

int main()
{
    char *infile="vao.txt", *outfile="ra.txt";
```

```

int i, j;
FILE *f1, *f2;
f1=fopen(infile,"rt");
if(f1==NULL)
{
    printf("Loi mo file %s",infile);
    getch();
    return -1;
}
f2=fopen(outfile,"wt");
if(f2==NULL)
{
    printf("Loi mo file %s",outfile);
    getch();
    fclose(f1);
    return -2;
}
fscanf(f1,"%d%d",&m,&n);
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        fscanf(f1,"%d",&a[i][j]);
    }
}
fprintf(f2,"%d %d\n",n,m);
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
    {
        fprintf(f2,"%4d",a[j][i]);
    }
    fprintf(f2,"\n");
}
fclose(f1);
fclose(f2);
printf("Ket thuc tinh chuyen vi");
getch();
return 0;
}

```

IV. Truy cập tập tin nhị phân

1. Ghi dữ liệu lên tập tin nhị phân

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.

2. Đọc dữ liệu từ tập tin nhị phân

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..

3. Ví dụ

Ví 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>
int main()
{
    FILE *f;
    f=fopen("number.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);
    }
    else
    {
        printf("Khong mo duoc file");
    }
    getch();
    return 0;
}
```

Ví 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.

Trong trường hợp này có thể coi file nhị phân như là nơi lưu trữ dữ liệu lâu dài, cũng có thể coi như là nơi lưu trữ xử lý dữ liệu thay vì dùng bộ nhớ.

```
#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");
    printf("Nhap bao nhieu sinh vien? ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("Sinh vien thu %i\n",i);
        fflush(stdin);
        printf(" - MSSV: ");gets(sv.Ma);
        printf(" - Ho ten: ");gets(sv.HoTen);
        fwrite(&sv,sizeof(sv),1,f);
    }
    fclose(f);
    printf("Bam phim bat ky de tiep tuc");
    getch();
}
void ReadFile(char *FileName)
{
    FILE *f;
    SinhVien sv;
    f=fopen(FileName,"rb");
    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=0;
    SinhVien sv;
    fflush(stdin);
    printf("Ma so sinh vien can tim: ");gets(MSSV);
    f=fopen(FileName,"rb");
    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();
}
int 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);
        fflush(stdin);

        if(c==1)
        {
            WriteFile("SinhVien.Dat");
        }
        else if (c==2)
        {
            ReadFile("SinhVien.Dat");
        }
    }
}

```

```

else if (c==3)
{
    Search("SinhVien.Dat");
}
else break;
}
return 0;
}

```

Ngoài ra thư viện `stdio.h` còn định nghĩa một số hàm khác cho phép thao tác với tập tin, sinh viên có thể tham khảo trong phần trợ giúp.

V. Tóm tắt nội dung bài học

- I. Một số khái niệm
- II. Các thao tác trên tập tin
 1. Khai báo biến tập tin
 2. Mở tập tin
 3. Đóng tập tin
 4. Kiểm tra đến cuối tập tin hay chưa?
 5. Di chuyển con trỏ tập tin về đầu tập tin - Hàm `rewind()`
- III. Truy cập tập tin văn bản
 1. Ghi dữ liệu lên tập tin văn bản
 2. Đọc dữ liệu từ tập tin văn bản
- IV. Truy cập tập tin nhị phân
 1. Ghi dữ liệu lên tập tin nhị phân - Hàm `fwrite()`
 2. Đọc dữ liệu từ tập tin nhị phân - Hàm `fread()`
 3. Di chuyển con trỏ tập tin - Hàm `fseek()`
 4. Ví dụ
- V. Tóm tắt nội dung bài học

VI. Bài tập

Xem **Bài 17 - Bài tập thực hành** Làm việc với File và Thuật toán nâng cao

Bài 17 - Bài thực hành LÀM VIỆC VỚI FILE

I. Bài tập làm theo yêu cầu

1. Viết chương trình tính nghiệm của hệ 2 phương trình với tham số nhập từ File

Vấn đề: Viết chương trình giải hệ 2 phương trình 2 ẩn sau:

$$a1.x + b1.y = c1$$

$$a2.x + b2.y = c2$$

với dữ liệu là các hệ số a1, b1, c1, a2, b2, c2 được nhập từ file week5_1.dat.

Soạn thảo văn bản chương trình sau:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a1, b1, c1;
    int a2, b2, c2;
    float d, dx, dy, x, y;
    FILE *finput;
    FILE *foutput;
    finput=fopen("week5_1.dat","rt");
    foutput=fopen("week5_1.res","wt");
    fscanf(finput,"%d %d %d",&a1, &b1, &c1);
    fscanf(finput,"%d %d %d",&a2, &b2, &c2);
    d=a1*b2-a2*b1;
    dx=c1*b2-c2*b1;
    dy=a1*c2-a2*c1;
    fprintf(foutput,"%s\n","He phuong trinh:");
    fprintf(foutput,"%d.x + %d.y = %d\n",a1,b1,c1);
    fprintf(foutput,"%d.x + %d.y = %d\n",a2,b2,c2);
    if (d!=0)
    {
        x = (float)dx/d; y = (float)dy/d;
        fprintf(foutput,"%s\n","Co cap nghiem:");
        fprintf(foutput,"x=%0.2f, y=%0.2f",x,y);
    }
    else
    {
        fprintf(foutput,"%s\n","Vo nghiem!");
    }
    fclose(finput);
    fclose(foutput);
    printf("Xem ket qua trong file week5_1.res");
    getch();
    return 0;
}
```

Thử nghiệm 1:

- Sử dụng công cụ notepad soạn thảo nội dung như sau:

5 7 9

7 2 1

Và ghi vào file **week5_1.dat** (tên file là **week5_1** và phần mở rộng là **dat**) cùng thư mục lưu trữ chương trình nguồn.

- Chạy chương trình và cho biết nội dung file **week5_1.res**. Nhận xét về kết quả đạt được.

Thử nghiệm 2:

- Sử dụng công cụ notepad sửa lại nội dung file **week5_1.dat** như sau:

```
5 7 9
10 14 18
```

- Chạy chương trình và cho biết nội dung file **week5_1.res**. Nhận xét về kết quả đạt được.

Thử nghiệm 3:

- Sử dụng công cụ notepad sửa lại nội dung file **week5_1.dat** như sau:

```
5.0 7.0 9.0
7.0 2.0 1.0
```

Chạy chương trình và cho biết nội dung file **week5_1.res**. Nhận xét về kết quả đạt được.

2. Kiểm tra định dạng file

Vấn đề: Làm thế nào để biết một file thực sự là file chương trình dạng EXE hoặc là một file ảnh BMP? Cách thứ nhất có thể là kiểm tra phần mở rộng của tên file, tuy nhiên cách này không đáng tin cậy vì thực tế người sử dụng có thể tự đặt cho phần mở rộng của tên file. Cách thứ 2 là kiểm tra những dấu hiệu đặc biệt trong nội dung của file đó để đưa ra kết luận. Cách thứ 2 này là một trong những việc mà một con virus máy tính cần làm để lây nhiễm lên các file cần thiết và cũng là cách để một chương trình diệt virus có thể kiểm tra để xem 1 file có bị nhiễm virus hay không.

Soạn thảo văn bản chương trình sau:

```
#include<stdio.h>
#include<conio.h>
int main()
{
FILE *f;
f=fopen("notepad.exe","rb");
if (f!=NULL)
{
char a,b;
fread(&a,sizeof(char),1,f);
fread(&b,sizeof(char),1,f);
printf("%c %c",a,b);
fclose(f);
}
else
{
printf("Khong mo duoc file");
}
getch();
return 0;
}
```

Thử nghiệm 1:

- Copy file **notepad.exe** trong thư mục C:\windows\system32 đưa vào thư mục lưu giữ chương trình.
- Chạy chương trình và nhận xét về kết quả hiển thị trên màn hình.
- Đòi dòng lệnh

```
f=fopen("notepad.exe","rb");
```

Thành

```
f=fopen("calc.exe","rb");
```

- Copy file **calc.exe** trong thư mục C:\windows\system32 đưa vào thư mục lưu giữ chương trình.
- Chạy chương trình và nhận xét về kết quả hiển thị trên màn hình.

Thử nghiệm 2:

- Trong thư mục lưu giữ chương trình, chọn một file có phần mở rộng là EXE và đổi tên thành **myfile.exe**.
- Đổi dòng lệnh

```
f=fopen("calc.exe","rb");
```

Thành

```
f=fopen("myfile.exe","rb");
```
- Chạy chương trình và nhận xét về kết quả hiển thị trên màn hình.
- Qua kết quả của thử nghiệm 1 và thử nghiệm 2 có thể kết luận sơ bộ gì về file chương trình dạng EXE.

Thử nghiệm 3:

- Sử dụng công cụ Paint của windows, tạo một ảnh và ghi vào thư mục lưu trữ chương trình với định dạng file là **24-bit Bitmap** và tên file là **Anh1.bmp**.
- Đổi dòng lệnh

```
f=fopen("myfile.exe","rb");
```

Thành

```
f=fopen("Anh1.bmp","rb");
```
- Chạy chương trình và nhận xét về kết quả hiển thị trên màn hình.
- Tương tự như trên tạo 2 file ảnh Anh2.bmp và Anh3.bmp.
- Chạy chương trình và kết luận sơ bộ về dạng file này.

Thử nghiệm 4:

- Về thư mục lưu trữ chương trình, chạy file **notepad.exe**, chương trình này có hoạt động bình thường không?
- Soạn thảo chương trình sau:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    FILE *f;
    f=fopen("notepad.exe","r+b");
    if (f!=NULL)
    {
        char a = 'N';
        fwrite(&a,sizeof(char),1,f);
        fclose(f);
    }
    else
    {
        printf("Khong mo duoc file");
    }
    getch();
    return 0;
}
```

- Đoạn chương trình trên làm gì?
- Về thư mục lưu trữ chương trình, chạy file **notepad.exe**, chương trình này có hoạt động bình thường không? Vì sao?
- Sửa dòng lệnh **char a = 'N'**; trong chương trình trên thành **char a = 'M'**; sau đó nhấn F9 để chạy chương trình.

- Về thư mục lưu trữ chương trình, chạy file **notepad.exe**, chương trình này có hoạt động bình thường không? Vì sao?

II. Bài tập tự làm

- 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.
- 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.
- Viết chương trình đếm số từ và số dòng trong một tập tin văn bản.
- 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.
- 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 1, 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	SO LG	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

Bài 18 - MỘT SỐ VẤN ĐỀ MỞ RỘNG

Nội dung bài học

- I. Cây, Hàm băm
- II. Khởi động đồ họa
- III. Các hàm đồ họa
- IV. Xử lý văn bản trên màn hình đồ họa
- V. Hiệu ứng hoạt hình đồ họa

Việc hiển thị thông tin trên màn hình máy tính được thực hiện thông qua một vi mạch điều khiển màn hình. Khi màn hình ở chế độ văn bản (text mode) chúng ta có thể hiển thị thông tin lên màn hình bằng các lệnh: `printf()`, `putch()`, `putchar()`, ... Thông tin mà chúng ta cần đưa ra màn hình được chuyển tới vi mạch điều khiển màn hình dưới dạng mã kí tự ASCII. Vi mạch nói trên có nhiệm vụ đưa kí tự đó theo mẫu định sẵn ra màn hình ở vị trí được xác định bởi chương trình của chúng ta.

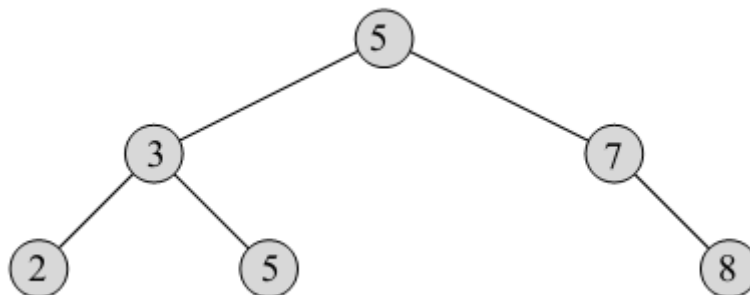
Ngoài chế độ văn bản, màn hình còn có thể làm việc trong chế độ đồ họa. Khi màn hình ở chế độ đồ họa chúng ta có thể vẽ đồ thị, viết chữ to hoặc thể hiện các hình ảnh khác - những việc mà chúng ta không thể thực hiện được trong chế độ văn bản.

Các hàm và thủ tục đồ họa được khai báo trong file `graphics.h`.

I. Cây, Hàm băm

1. Cây

- Danh sách phi tuyến là danh sách các đối tượng cần lưu trữ không theo mô hình dãy liệt kê liên tục. Trong thực tế nhiều kiểu dữ liệu không biểu diễn dưới dạng danh sách liên tiếp vì thế cần mô hình dữ liệu khác để mô tả các kiểu dữ liệu này. Ví dụ mô tả các cây tìm kiếm, đồ thị, ...
- Trong phần này sẽ trình bày cấu trúc dữ liệu động mô tả cây trong máy tính. Cây là cấu trúc dữ liệu chỉ các nút, mỗi nút có thể trở đến không hoặc nhiều nút khác (con) có cùng cấu trúc như nó. Bản thân định nghĩa câu là một định nghĩa đệ qui bởi chính một nút trở đến có thể nhiều nút khác, chính các nút được trở đến này cũng là một cây.



- Hình 1. Mô hình cây.

- Theo hình vẽ, nút (5) là một cây có hai con là (3), (7), nhưng bản thân (3), (7) cũng là một cây.

- Các phần tử trên có thể không có con (2), (5), (8), hoặc có nhiều con tùy thuộc vào mục tiêu thiết kế và mô tả của hệ thống. Trong bài này sẽ giới thiệu về mô hình cây có nhiều nhất là hai con. Mô hình cây nhị phân tìm kiếm như sau: mỗi đỉnh có nhiều nhất là hai con, các nút con thuộc cây trái luôn bé hơn đỉnh, và các nút của cây con phải luôn lớn hơn hoặc bằng đỉnh. Các thao tác với cây bao gồm khởi tạo – init(), thêm phần tử - appendlabel(), tìm kiếm phần tử trong cây – seach(), giải phóng cây – dispose(), và có thể có thêm một số thao tác khác.

- Khởi tạo cây
- Hàm init()
- Dữ liệu vào: con trỏ đến con trỏ cây

```
typedef struct trees{
    int label;
    trees *left, *right;
};
void init(trees ** top)
{
    *top=NULL;
}
```

- Thêm phần tử vào cây, đảm bảo theo qui tắc đã mô tả
- Hàm appendlabel()
- Dữ liệu vào: con trỏ đến con trỏ cây, phần tử cần thêm vào
- return: 1 nếu không thành công, 0 nếu thêm thành công

```
int append(trees **top, trees *temp)
{
    trees *t>(*top);
    if(t->label<temp->label)
    {
        if(t->left==NULL)
        {
            t->left=temp;
            return 0;
        }
        return append(&(t->left),temp);
    }
    else
    {
        if(t->right==NULL)
        {
            t->right=temp;
            return 0;
        }
        return append(&(t->right),temp);
    }
}
```

```

}
int appendlabel(trees ** top, int label)
{
    trees *temp;
    temp=(trees*)malloc(sizeof(trees));
    if(temp==NULL)
    {
        return 1;
    }
    temp->label=label;
    temp->left=NULL;
    temp->right=NULL;
    if(*top==NULL)
    {
        *top=temp;
        return 0;
    }
    return append(top,temp);
}

```

- - Tìm kiếm tra sự xuất hiện của phần tử trong cây
- Hàm search()
- - Dữ liệu vào: con trỏ đến con trỏ cây, giá trị cần tìm kiếm
- - return: con trỏ null nếu không tìm thấy, ngược lại con trỏ đến nút tìm thấy

```

trees * search(trees **top,int label)
{
    if(*top==NULL)
    {
        return NULL;
    }
    printf("Tim kiem cho %d ",(*top)->label);
    if((*top)->label==label)
    {
        return (*top);
    }
    if((*top)->label<label)
    {
        printf(" --> trai\n");
        return search(&((*top)->left),label);
    }
    else
    {
        printf(" --> phai\n");
        return search(&((*top)->right),label);
    }
}

```

- - Giải phóng toàn bộ các phần tử của cây
- Hàm dispose()

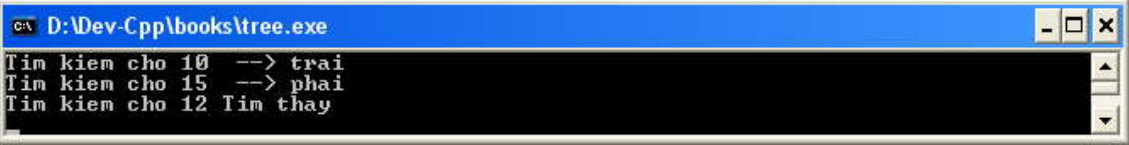
- Dữ liệu vào: con trỏ đến con trỏ đỉnh của cây

```
void dispose(trees **top)
{
    if(*top==NULL)
    {
        return ;
    }
    dispose(&((*top)->left));
    dispose(&((*top)->right));
    delete(*top);
    *top=NULL;
}
```

- Chương trình thử nghiệm

```
int main(int argc, char* argv[])
{
    trees *top, *t;
    init(&top);
    appendlabel(&top,10);
    appendlabel(&top,7);
    appendlabel(&top,15);
    appendlabel(&top,12);
    appendlabel(&top,18);
    appendlabel(&top,6);
    appendlabel(&top,9);
    t=search(&top,12);
    if(t!=NULL)
    {
        printf("Tim thay\n");
    }
    else
    {
        printf("Khong tim thay\n");
    }
    dispose(&top);
    getch();
    return 0;
}
```

- Kết quả chương trình



```
c:\ D:\Dev-Cpp\books\tree.exe
Tim kien cho 10 --> trai
Tim kien cho 15 --> phai
Tim kien cho 12 Tim thay
```

2. Hàm băm

Tham khảo: http://vi.wikipedia.org/wiki/H%C3%A0m_b%C4%83m

Hàm băm (tiếng Anh: hash function) là giải thuật nhằm sinh ra các giá trị băm tương ứng với mỗi khối dữ liệu (có thể là một chuỗi kí tự, một đối tượng trong lập trình hướng đối tượng, v.v...). Giá trị băm đóng vai gần như một khóa để phân biệt các khối dữ liệu, tuy nhiên, người ta chấp hiện tượng trùng khóa hay còn gọi là đụng độ và cố gắng cải thiện giải thuật để giảm thiểu sự đụng độ đó. Hàm băm thường được dùng trong bảng băm nhằm giảm chi phí tính toán khi tìm

một khối dữ liệu trong một tập hợp (nhờ việc so sánh các giá trị băm nhanh hơn việc so sánh những khối dữ liệu có kích thước lớn).

Vì tính thông dụng của bảng băm, ngày nay, đa số ngôn ngữ lập trình đều cung cấp thư viện ứng dụng bảng băm, thường gọi là thư viện collection trong đó có các vấn đề như: tập hợp (collection), danh sách (list), bảng (table), ánh xạ (mapping), từ điển (dictionary)). Thông thường, các lập trình viên chỉ cần viết hàm băm cho các đối tượng nhằm tích hợp với thư viện bảng băm đã được xây dựng sẵn.

Một hàm băm tốt phải thỏa mãn các điều kiện sau:

- Tính toán nhanh.
- Các khoá được phân bố đều trong bảng.
- Ít xảy ra đụng độ.
- Xử lý được các loại khóa có kiểu dữ liệu khác nhau

Xem thêm tài liệu tham khảo, internet để biết chi tiết.

II. Khởi động đồ họa

Chuyển chế độ màn hình về chế độ đồ họa cho phép thực hiện các lệnh vẽ đối tượng đồ họa.

Cú pháp: `void initgraph(int *graphdriver,int graphmode,char *driverpath);`

Trong đó:

- + driverpath: là xâu ký tự chỉ đường dẫn đến thư mục chứa các tập tin điều khiển đồ họa.
- + graphdriver: cho biết màn hình đồ họa sử dụng trong chương trình.
- + graphmode: cho biết mode đồ họa sử dụng trong chương trình.

Bảng 13-1

Các giá trị có thể của graphdriver và graphmode

Graphdriver	graphmode	Độ phân giải
CGA (1)	CGAC0 (0)	320x200
	CGAC1 (1)	320x200
	CGAC2 (2)	320x200
	CGAC3 (3)	320x200
	CGAHI (4)	640x200
MCGA (2)	MCGA0 (0)	320x200
	MCGA1 (1)	320x200
	MCGA2 (2)	320x200
	MCGA3 (3)	320x200
	MCGAMed (4)	640x200
MCGAHI (5)	MCGAHI (5)	640x480
EGA (3)	EGAL0 (0)	640x200
	EGAHI (1)	640x350
EGA64 (4)	EGA64LO (0)	640x200
	EGA64HI (1)	640x350
EGAMONO (5)	EGAMONOH (0)	640x350
VGA (9)	VGALO (0)	640x200

	VGAMED (1)	640x350
	VGAHI (2)	640x480
HERCMONO (7)	HERCMONOHI	720x348
ATT400 (8)	ATT400C0 (0)	320x200
	ATT400C1 (1)	320x200
	ATT400C2 (2)	320x200
	ATT400C3 (3)	320x200
	ATT400MED (4)	640x400
	ATT400HI (5)	640x400
PC3270 (10)	PC3270HI (0)	720x350
IBM8514 (6)	PC3270LO (0)	640x480 256 màu
	PC3270HI (1)	1024x768 256 màu

Bảng trên cho thấy độ phân giải của màn hình phụ thuộc cả vào kiểu màn hình và mode. Ví dụ như trong màn hình EGA nếu dùng EGALO thì độ phân giải là 640x200 (Hàm getmaxx() cho giá trị cực đại của số điểm theo chiều ngang của màn hình. Hàm getmaxy() cho giá trị cực đại của số điểm theo chiều dọc của màn hình.).

Ví dụ: Khởi tạo đồ họa trong môi trường lập trình Turbo C. Giả sử máy tính có màn hình VGA, các tập tin đồ họa chứa trong thư mục C:\TC\BGI, khi đó ta khởi động chế độ đồ họa cho màn hình như sau:

```
#include <graphics.h>
main()
{
    int mh=VGA,mode=VGAHI; /*Hoặc mh=9,mode=2*/
    initgraph(&mh,&mode,"C:\\TC\\BGI");
}
```

Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver bằng DETECT hay giá trị 0. Khi đó, kết quả của initgraph sẽ là:

Kiểu màn hình đang sử dụng được phát hiện, giá trị của nó được gán cho biến graphdriver.

Mode đồ họa ở độ phân giải cao nhất ứng với màn hình đang sử dụng cũng được phát hiện và giá trị của nó được gán cho biến graphmode.

Như vậy dùng hằng số DETECT chẳng những có thể khởi tạo được chế độ đồ họa cho màn hình hiện có theo mode có độ phân giải cao nhất mà còn giúp ta xác định kiểu màn hình đang sử dụng.

Ví dụ khởi tạo đồ họa trong môi trường DevC++.

Kiểm tra tính tồn tại của file tiêu đề thư viện [graphics.h](#) trong thư mục include, và thư viện [libbgi.a](#) lib.

Tạo một project trong cửa sổ option của project chọn tab Parameters chọn các file thư viện:

```
-libbgi
-libgdi32
-libcomdlg32
-libuuid
-liboleaut32
-libole32
```

Khởi tạo giống như trong Turbo C, với tham số cuối cùng là chuỗi trống.

```
#include <graphics.h>
main()
{
    int mh=VGA,mode=VGAHI; /*Hoặc mh=9,mode=2*/
    initgraph(&mh,&mode,"");
}
```

Ví dụ: Chương trình dưới đây xác định kiểu màn hình đang sử dụng:

```
#include <graphics.h>
#include <stdio.h>
main()
{
    int mh=0, mode;
    initgraph(&mh,&mode,"C:\\TC\\BGI");
    closegraph();
    printf("\n Gia tri so cua man hinh la: %d",mh);
    printf("\n Gia tri so mode do hoa la: %d",mode);

    getch();
}
```

Nếu chuỗi dùng để xác định driverpath là chuỗi rỗng thì chương trình dịch sẽ tìm kiếm các file điều khiển đồ họa trên thư mục hiện thời và thư của của trình biên dịch.

Chú ý: Màn hình đồ họa máy tính được xác định tọa độ theo chiều từ trái sang phải và từ trên xuống dưới khác với tọa độ không gian decac.



Vì thế khi vẽ hình trên màn hình cần chuyển trục tọa độ về vị trí phù hợp, và đảo trục tọa độ theo tọa độ thực.

III. Các hàm đồ họa

1. Mẫu và màu

- **Đặt màu nền:** đặt màu cho toàn bộ nền trong Turbo C, trong DevC++ đặt màu nền cho các lệnh có vẽ nền ở sau.

Cú pháp: *void setbkcolor(int cl);*

+ cl: là màu cần đặt, xem bảng màu ở dưới

trong DevC++ có thể sử dụng thêm lệnh bar để điền toàn bộ màn hình với màu nền.

- **Đặt màu đường vẽ:** Để đặt màu vẽ đường trong các lệnh vẽ sau đó.

Cú pháp: *void setcolor(int cl);*

+ cl: màu cần đặt, xem bảng màu ở dưới

- **Đặt mẫu (kiểu) tô và màu tô:** Để đặt mẫu (kiểu) tô và màu tô ta dùng thủ tục có điền nền.

Cú pháp: *void setfillstyle(int par, int cl);*

+ pr: kiểu điền, theo bảng ở dưới

+ cl: kiểu màu, theo bảng ở dưới.

Bảng 13-2

Các giá trị màu

Tên hằng	Giá trị số	Màu hiển thị
BLACK	0	Đen
BLUE	1	Xanh da trời
GREEN	2	Xanh lá cây
CYAN	3	Xanh lơ
RED	4	Đỏ
MAGENTA	5	Tím
BROWN	6	Nâu
LIGHTGRAY	7	Xám nhạt
DARKGRAY	8	Xám đậm
LIGHTBLUE	9	Xanh xa trời nhạt
LIGHTGREEN	10	Xanh lá cây nhạt
LIGHTCYAN	11	Xanh lơ nhạt
LIGHTRED	12	Đỏ nhạt
LIGHTMAGEN	13	Tím nhạt
TA		
YELLOW	14	Vàng
WHITE	15	Trắng

Các kiểu mẫu tô

Tên hằng	Giá trị số	Kiểu mẫu tô
EMPTY_FILL	0	Tô bằng màu nền
SOLID_FILL	1	Tô bằng đường liền nét
LINE_FILL	2	Tô bằng đường -----
LTSLASH_FILL	3	Tô bằng ///
SLASH_FILL	4	Tô bằng /// in đậm
BKSLASH_FILL	5	Tô bằng \\ in đậm
LTBKSLASH_FILL	6	Tô bằng \\\
HATCH_FILL	7	Tô bằng đường gạch bóng nhạt
XHATCH_FILL	8	Tô bằng đường gạch bóng chữ thập
INTERLEAVE_FIL L	9	Tô bằng đường đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm thưa
CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

- **Chọn lại giá trị màu cho màu cơ bản:** 16 màu cơ bản được chọn từ danh sách 256 màu. Hệ thống cho người dùng tự chọn lại màu cơ bản từ 256 màu.

Cú pháp: void setpalette(int clOr, int cl);

+ clOr: vị trí của màu trong bảng màu cơ bản

+ cl: Màu được đưa vào theo danh sách 256;

Ví dụ: Câu lệnh: setpalette(0,lightcyan); đổi màu đầu tiên trong bảng màu thành màu xanh lơ nhạt. Các màu khác không bị ảnh hưởng. Lệnh này không thực sự có hiệu quả trong DevC++;

- **Lấy bảng màu hiện thời:**

- Lấy màu vẽ hiện thời, trả về màu đã xác định bằng thủ tục setcolor ngay trước nó.

Cú pháp: int getcolor()

- Lấy màu nền hiện thời, trả về màu nền đã đặt bởi hàm setbkcolor trước đó.

Cú pháp: int getbkcolor()

2. Vẽ và tô màu đường tròn

Có thể chia các đường và hình thành bốn nhóm chính:

- Cung tròn và hình tròn.

- Đường gấp khúc và đa giác.
- Đường thẳng.
- Hình chữ nhật.

Cung tròn và đường tròn

Nhóm này bao gồm: Cung tròn, đường tròn, cung elip và hình quạt.

- Cung tròn:** Để vẽ một cung tròn

Cú pháp: *void arc(int x, int y, int gd, int gc, int r);*

- + (x,y): là toạ độ tâm cung tròn.
- + gd: là góc đầu cung tròn(0 đến 360 độ).
- + gc: là góc cuối cung tròn (gd đến 360 độ).
- + r: là bán kính cung tròn .

Ví dụ:

Vẽ một cung tròn có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán kính 30.
`arc(100,50,0,180,30);`

- Đường tròn:** Để vẽ đường tròn

Cú pháp: *void circle(int x, int y, int r);*

- + (x,y) : là toạ độ tâm cung tròn.
- + r : là bán kính đường tròn.

Ví dụ:

Vẽ một đường tròn có tâm tại (100,50) và bán kính 30.
`circle(100,50,30);`

- Cung elip:** Để vẽ một cung elip

Cú pháp: *void ellipse(int x, int y, int gd, int gc, int xr, int yr);*

- + (x,y) : là toạ độ tâm cung elip.
- + gd : là góc đầu cung tròn(0 đến 360 độ).
- + gc : là góc cuối cung tròn (gd đến 360 độ).
- + xr : là bán trục nằm ngang.
- + yr : là bán trục thẳng đứng.

Ví dụ:

Vẽ một cung elip có tâm tại (100,50), góc đầu là 0, góc cuối là 180, bán trục ngang 30, bán trục đứng là 20.
`ellipse(100,50,0,180,30,20);`

- Hình quạt:** Để vẽ và tô màu một hình quạt

Cú pháp: *void pieslice(int x, int y, int gd, int gc, int r);*

- + (x,y) : là toạ độ tâm hình quạt.
- + gd : là góc đầu hình quạt (0 đến 360 độ).
- + gc : là góc cuối hình quạt (gd đến 360 độ).

+ r: là bán kính hình quạt .

Ví dụ: Chương trình dưới đây sẽ vẽ một cung tròn ở góc phần tư thứ nhất, một cung elip ở góc phần tư thứ ba, một đường tròn và một hình quạt quét từ 90 đến 360 độ.

```
# include "graphics.h"
#include "stdio.h"
#include "conio.h"
main()
{
    int md=0,mode;
    initgraph(&md,&mode,"C:\\TC\\BGI");
    setbkcolor(BLUE);
    setcolor(YELLOW);
    setfillstyle(SOLID_FILL,RED);;
    arc(160,50,0,90,45);
    circle(160,150,45);
    pieslice(480,150,90,360,45);
    getch();
    closegraph();
}
```

3. Vẽ đường gấp khúc và đa giác

- **Vẽ đường gấp khúc:** Muốn vẽ đường gấp khúc đi qua n điểm: (x_1,y_1) , (x_2,y_2) , ..., (x_n,y_n) các tọa độ (x_i,y_i) được gán cho một mảng a kiểu int theo nguyên tắc sau:

Toạ độ x_1 gán cho $a[0]$
Toạ độ y_1 gán cho $a[1]$
Toạ độ x_2 gán cho $a[2]$
Toạ độ y_2 gán cho $a[3]$
....
Toạ độ x_n gán cho $a[2n-2]$
Toạ độ y_n gán cho $a[2n-1]$

Sau đó gọi hàm:

Cú pháp: *void drawpoly(n,a);*

n: số điểm cần vẽ tương ứng số lượng điểm được lưu trữ trong a

a: Các điểm trên đường gấp khúc được lưu trữ theo mô hình trên

Nếu điểm cuối cùng (x_n,y_n) trùng với điểm đầu (x_1,y_1) thì đường gấp khúc khép kín.

Ví dụ vẽ tam giác

```
int a[8]={10,10,100,100,10,100,10,10};
drawpoly(4,a);
```

- **Tô màu đa giác:** Vẽ đa giác và tô màu cho phần đa giác đã vẽ

Cú pháp: *void fillpoly(n,a);*

n: số điểm cần vẽ

a: danh sách các điểm cần vẽ

sẽ vẽ và tô màu một đa giác có đỉnh là các điểm $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$

Ví dụ: Vẽ một đường gấp khúc và hai đường tam giác.

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
int poly1[]={5,200,190,5,100,300};
int poly2[]={205,200,390,5,300,300};
int poly3[]={405,200,590,5,500,300,405,200};
main()
{
    int md=0,mode;
    initgraph(&md,&mode,"C:\\TC\\BGI");
    setbkcolor(CYAN);
    setcolor(YELLOW);
    setfillstyle(SOLID_FILL,MAGENTA);
    drawpoly(3,poly1);
    fillpoly(3,poly2);
    fillpoly(4,poly3);
    getch();
    closegraph();
}
```

Vẽ đường thẳng: đường thẳng nối hai điểm

Cú pháp: *void line(int x1, int y1, int x2, int y2);*

+ (x_1, y_1) : tọa độ điểm 1

+ (x_2, y_2) : tọa độ điểm 2

Vẽ đường thẳng: đường thẳng từ điểm con trỏ đồ họa hiện tại đến điểm mới

Cú pháp: *void lineto(int x, int y);*

(x, y) : điểm sẽ vẽ đến từ con trỏ đồ họa hiện tại

Vẽ đường thẳng: đường thẳng từ điểm con trỏ đồ họa hiện tại đến điểm cách điểm hiện tại một khoảng

Cú pháp: *void linerel(int dx, int dy);*

(dx, dy) : khoảng cách với con trỏ đồ họa hiện tại, nếu điểm hiện tại là (x, y) thì điểm mới cần vẽ là $(x+dx, y+dy)$

Di chuyển con chạy đồ họa: Để di chuyển con chạy đến vị trí mới

Cú pháp: *void moveto(int x, int y);*

+ (x, y) : điểm con trỏ đồ họa mới

Ví dụ:

```
moveto(200,100);
line(100,100,1,1);
lineto(100,200);
linerel(100,100);
```

Chọn kiểu đường: kiểu đường để vẽ các đường thẳng, đa giác, ...

Cú pháp : void setlinestyle(int linestyle, int par, int thin);

+ linestyle : kiểu đường, được miêu tả ở dưới

+ par : nếu tham số thứ nhất là USERBIT_LINE thì tuân thủ theo bit của tham số này để tạo ra điểm vẽ.

+ thin : độ dày của đường được miêu tả ở dưới

Bảng 13-4

Mô tả kiểu đường		
Tên hằng	Giá trị số	Kiểu đường
SOLID_LINE	0	Nét liền
DOTTED_LINE	1	Nét chấm
CENTER_LINE	2	Nét chấm gạch
DASHED_LINE	3	Nét gạch
USERBIT_LINE	4	Mẫu tự tạo

Bảng 13-5

Mô tả độ dày của đường		
Tên hằng	Giá trị số	Bề dày
NORM_WIDTH	1	Bề dày bình thường
THICK_WIDTH	3	Bề dày gấp ba

Ví dụ: Chương trình vẽ một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh sau:

```
(20,20),(620,20),(620,180),(20,180) và (320,100)
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
main()
{
    int mh=0, mode;
    initgraph(&mh,&mode,"C:\\TC\\BGI");
    setbkcolor(BLUE);
    setcolor(YELLOW);
    setlinestyle(SOLID-LINE,0,THICK_WIDTH);
    moveto(320,100); /* con ch?y ? v? trí ( 320,100 ) */
    line(20,20,620,20); /* con ch?y v?n ? v? trí ( 320,100 ) */
    linerel(-300,80);
    lineto(620,180);
    lineto(620,20);
    getch();
    closegraph();
}
```

4. Vẽ điểm, miền

- **Vẽ điểm:** vẽ điểm ảnh trên màn hình với màu xác định

Cú pháp: *void putpixel(int x, int y, int color);*

+ (x,y): tọa độ điểm cần vẽ

+ color: màu vẽ

- **Lấy màu điểm vẽ:** xác định màu của một điểm trên màn hình

Cú pháp: *unsigned getpixel(int x, int y);*

+ (x,y): Tọa độ của điểm cần lấy màu

+ giá trị trả về là màu của điểm trên màn hình

- **Tô miền:** Tô màu cho một miền kín được xác định bởi màu biên, nếu vùng không kín sẽ tô màu hết vùng làm việc.

Cú pháp: *void floodfill(int x, int y, int border);*

+ (x,y): tọa độ xác định điểm tô màu đầu tiên

+ border: màu của vùng biên

Ví dụ: Vẽ một đường tròn màu đỏ trên màn hình màu xanh. Tọa độ (x,y) của điểm gieo được nạp từ bàn phím. Tùy thuộc giá trị cụ thể của x,y chương trình sẽ tô màu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include "graphics.h"
#include "stdio.h"
#include <conio.h>
main()
{
    int mh=0,mode=0, x, y;
    printf("\nVao toa do x,y:");
    scanf("%d%d",&x,&y);
    initgraph(&mh,&mode,"");
    setbkcolor(BLUE);
    setcolor(RED);
    setfillstyle(11,YELLOW);
    circle(320,100,50);
    moveto(1,150);
    floodfill(x,y,RED);
    getch();
    closegraph();
}
```

5. Hình chữ nhật

- **Vẽ hình chữ nhật:** xác định hai đỉnh là góc trái trên cùng, và góc phải dưới cùng của hình chữ nhật

Cú pháp: *void rectangle(int x1, int y1, int x2, int y2);*

+ (x1,y1) : tọa độ góc trái, phía trên hình chữ nhật

+ (x2,y2) : tọa độ góc phải phía dưới hình chữ nhật

- **Vẽ hình chữ nhật được tô kín:** vẽ hình chữ nhật với miền trong được tô màu

Cú pháp : `void bar(int x1, int y1, int x2, int y2);`

+ (x1,y1) : tọa độ góc trái, phía trên hình chữ nhật

+ (x2,y2) : tọa độ góc phải phía dưới hình chữ nhật

- **Vẽ hình chữ nhật có tô bóng:** vẽ hình chữ nhật có tô bóng 3D.

Cú pháp: `void bar3d(int x1, int y1, int x2, int y2, int depth, int top);`

+ (x1,y1) : tọa độ góc trái, phía trên hình chữ nhật

+ (x2,y2) : tọa độ góc phải phía dưới hình chữ nhật

+ depth: độ sâu của tô bóng

+ top: vẽ đỉnh (0: không vẽ, 1: vẽ)

Ví dụ: Chương trình dưới đây tạo nên một hình chữ nhật, một khối hình chữ nhật và một hình hộp có nắp:

```
#include "graphics.h"
main()
{
    int mh=0,mode=0;
    initgraph(&mh,&mode,"");
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(CLOSE_DOT_FILL,YELLOW);
    rectangle(5,5,300,160);
    bar(3,175,300,340);
    bar3d(320,100,500,340,100,1);
    getch();
    closegraph();
}
```

6. Cửa sổ (Viewport)

- **Thiết lập viewport:** tạo cửa sổ làm việc ảo trên màn hình.

Cú pháp: `void setviewport(int x1, int y1, int x2, int y2, int clip);`

+ (x1,y1) : tọa độ góc trên bên trái

+ (x2,y2) : tọa độ góc dưới bên phải

+ clip : 0 : cho phép vẽ ra ngoài viewport, 1 : không cho phép vẽ ra ngoài viewport

Ví dụ:

```
setviewport(100,50,200,150,0);
line(-20,-20,50,50);
```

Lập nên một vùng viewport hình chữ nhật có tọa độ góc trái cao là (100,50) và tọa độ góc phải thấp là (200,150) (là tọa độ trước khi đặt viewport).

Chú ý: Sau khi lập viewport, ta có hệ tọa độ mới mà góc trên bên trái sẽ có tọa độ (0,0).

- **Nhận diện viewport hiện hành:** thông tin về viewport hiện hành.

Cú pháp : `void getviewsetting(struct viewporttype *vp);`

vp: thông tin về viewport hiện thời,

cấu trúc của viewport

```
struct viewporttype
{
    int left,top,right,bottom;
    int clip;
};
```

- **Xóa viewport:** xóa các đối tượng vẽ trong viewport

Cú pháp: *void clearviewport(void);*

- **Xoá màn hình, đưa con chạy về tạo độ (0,0) của màn hình:**

Cú pháp: *void cleardevice(void);*

- **Toạ độ âm dương:**

Với chế độ cho phép vẽ ra ngoài viewport, các hàm đồ họa có thể vẽ ra ngoài viewport với tọa độ âm.

```
int xc,yc;
xc=getmaxx()/2;
yc=getmaxy()/2;
setviewport(xc,yc,getmaxx(),getmaxy(),0);
```

Như thế, màn hình sẽ được chia làm bốn phần với tọa độ âm dương như sau:

Phần tư trái trên: x âm, y âm.

x: từ -getmaxx()/2 đến 0.

y: từ -getmaxy()/2 đến 0.

Phần tư trái dưới: x âm, y dương.

x: từ -getmaxx()/2 đến 0.

y: từ 0 đến getmaxy()/2.

Phần tư phải trên: x dương, y âm.

x: từ 0 đến getmaxx()/2.

y: từ -getmaxy()/2 đến 0.

Phần tư phải dưới: x dương, y dương.

x: từ 0 đến getmaxx()/2.

y: từ 0 đến getmaxy()/2.

Ví dụ: Chương trình vẽ đồ thị hàm sin x trong hệ trục tọa độ âm dương. Hoàn hảo x lấy các giá trị từ -4π đến 4π . Trong chương trình có sử dụng hai hàm mới là `settextjustify` và `outtextxy` ta sẽ đề cập ngay trong phần sau.

```
#include "graphics.h"
#include "conio.h"
#include "math.h"
```



```

#define TYLEX 20
#define TYLEY 60
main()
{
    int mh=0,mode=DETECT;
    int x,y,i;
    initgraph(&mh,&mode,"");
    setviewport(getmaxx()/2,getmaxy()/2,getmaxx(),getmaxy(),0);
    setbkcolor(BLUE);
    setcolor(YELLOW);
    line(-getmaxx()/2,0,getmaxx()/2,0);
    line(0,-getmaxy()/2,0,getmaxy()/2);
    settxtjustify(1,1);
    setcolor(WHITE);
    outtextxy(0,0,"(0,0)");
    for (i=-400;i<=400;++i)
        {
            x=floor(2*M_PI*i*TYLEX/200);
            y=floor(sin(2*M_PI*i/200)*TYLEY);
            putpixel(x,y,WHITE);
        }
    getch();
    closegraph();
}

```

IV. Xử lý văn bản trên màn hình đồ họa

1. Hiện thị văn bản trên màn hình đồ họa

Cú pháp: *void outtext(char *s);*

s: chuỗi văn bản được hiển thị tại vị trí con trỏ đồ họa hiện tại

Cú pháp: *void outtextxy(int x, int y, char *s);*

(x,y): tọa độ sẽ hiển thị chuỗi

s: chuỗi văn bản sẽ được hiển thị

Ví dụ:

Hai cách viết dưới đây:

```
outtextxy(50,50," Say HELLO");
```

và

```
moveto(50,50);
outtext(" Say HELLO");
```

cho cùng kết quả.

2. Sử dụng các Fonts chữ

Các Fonts chữ nằm trong các tập tin *.CHR trên đĩa. Các Fonts này cho các kích thước và kiểu chữ khác nhau, chúng sẽ được hiển thị lên màn hình bằng các hàm outtext và outtextxy.

Để chọn và nạp Fonts ta dùng hàm:

Cú pháp : *void settextstyle(int font, int direction, int charsize);*

+ font : lựa chọn kiểu chữ và nhận một trong các hằng sau:

```
DEFAULT_FONT=0
TRIPLEX_FONT=1
SMALL_FONT=2
SANS_SERIF_FONT=3
GOTHIC_FONT=4
```

+ direction: để chọn hướng chữ và nhận một trong các hằng sau:

```
HORIZ_DIR=0    văn bản hiển thị theo hướng nằm ngang từ trái qua phải.
VERT_DIR=1     văn bản hiển thị theo hướng thẳng đứng từ dưới lên trên.
```

+ Charsize: là hệ số phóng to của ký tự và có giá trị trong khoảng từ 1 đến 10.

```
Khi charsize=1, font hiển thị trong hình chữ nhật 8*8 pixel.
Khi charsize=2 font hiển thị trong hình chữ nhật 16*16 pixel.
```

.....

```
Khi charsize=10, font hiển thị trong hình chữ nhật 80*80 pixel.
```

Các giá trị do `settextstyle` lập ra sẽ giữ nguyên tới khi gọi một `settextstyle` mới.

Ví dụ:

Các dòng lệnh:

```
settextstyle(3,1,3);//DevC++ không chuyển thẳng đứng
outtextxy(30,30,"GODS TRUST YOU");
```

sẽ hiển thị tại vị trí (30,30) dòng chữ GODS TRUST YOU theo chiều từ dưới lên trên, font chữ chọn là SANS_SERIF_FONT và cỡ chữ là 2.

□ **Đặt vị trí hiển thị của các xâu ký tự cho bởi `outtext` và `outtextxy`:**

Cú pháp: `void settextjustify(int horiz, int vert);`

+ horiz: có thể là một trong các hằng số sau:

```
LEFT_TEXT=0    ( Văn bản xuất hiện bên phải con chạy).
CENTER_TEXT    ( Chính tâm văn bản theo vị trí con chạy).
RIGHT_TEXT     (Văn bản xuất hiện bên trái con chạy).
```

+ vert: có thể là một trong các hằng số sau:

```
BOTTOM_TEXT=0 ( Văn bản xuất hiện phía trên con chạy).
CENTER_TEXT=1 ( Chính tâm văn bản theo vị trí con chạy).
TOP_TEXT=2    ( Văn bản xuất hiện phía dưới con chạy).
```

Ví dụ:

```
settextjustify(1,1);
outtextxy(100,100,"ABC");
```

3. Bề rộng và chiều cao của ký tự

- Chiều cao của chuỗi ký tự

Cú pháp: `int textheight(char *s);`

+ s: chuỗi ký tự cần kiểm tra

+ trả về độ cao của chuỗi ký tự khi in ra màn hình tính bằng pixel.

Ví dụ 1:

Với font bit map và hệ số phóng đại là 1 thì `textheight("A")` có giá trị là 8.

Ví dụ 2:

```
#include "stdio.h"
#include "graphics.h"
main()
{
    int mh=0,mode=DETECT, y,size;
    initgraph(&mh,&mode,"C:\\TC\\BGI");
    y=10;
    setttextjustify(0,0);
    for (size=1;size<5;++size)
        {
            setttextstyle(0,0,size);
            outtextxy(0,y,"SACRIFICE");
            y+=textheight("SACRIFICE")+10;
        }
    getch();
    closegraph();
}
```

- Bề rộng của ký tự:

Cú pháp: `int textwidth(char *s);`

+ s: chuỗi ký tự cần kiểm tra

+ trả về độ rộng của chuỗi ký tự khi in lên màn hình

V. Hiệu ứng hoạt hình đồ họa

Hiệu ứng hoạt hình là hiệu ứng các đối tượng di chuyển trên màn hình. Bản chất của việc dịch chuyển trên màn hình là xóa vị trí cũ và vẽ đối tượng sang vị trí mới.

- Xóa vị trí cũ bằng cách vẽ đối tượng với màu của màu nền

Ví dụ vẽ quả bóng di chuyển từ góc trên trái xuống góc phải dưới

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
#include <dos.h>
int draw(int x, int y, int color)
{
    setcolor(color);
    setbkcolor(color);
    setfillstyle(0,color);
    pieslice(x,y,0,360,10);
    return 0;
}
main()
{
    int i,x,y;
    int dx=5;
    int dy=5;
    int md=0,mode;
```

```

initgraph(&md,&mode,"C:\\TC\\BGI");
x=10;
y=10;
while(!kbhit())
{
    draw(x,y,15);
    delay(10);
    draw(x,y,0);
    x+=dx;
    y+=dy;
    if(y<10 || y>300)
        dy=-dy;
    if(x<10 || x>400)
        dx=-dx;
}
getch();
closegraph();
}

```

- Trong tình huống nền có hình ảnh khác việc vẽ lại màu nền sẽ xóa các đối tượng trên nền. Để giải quyết vấn đề này cần phải lưu trữ lại tình trạng màn hình trước khi vẽ, sau đó khôi phục lại màn hình để tiếp tục vẽ sang vị trí mới.

```

#include "graphics.h"
#include "stdio.h"
#include "conio.h"
#include <dos.h>
#define r 5
char *b;
int draw(int x, int y, int color)
{
    setcolor(color);
    setbkcolor(color);
    setfillstyle(0,color);
    pieslice(x,y,0,360,r);
    return 0;
}
int main()
{
    int i,x,y;
    int dx=5;
    int dy=5;
    delay(10);
    int md=0,mode=0;
    initgraph(&md,&mode,"C:\\TC\\BGI");
    b=(char*)malloc(imagesize(1,1,2*r+1,2*r+1));
    if(b==NULL)
    {
        closegraph();
        return -1;
    }
    line(0,0,getmaxx(),getmaxy());
    line(0,getmaxy(),getmaxx(),0);
    x=10;
    y=10;

```

```

while(!kbhit())
{
    getimage(x-r,y-r,x+r,y+r,b);
    draw(x,y,15);
    delay(10);
    putimage(x-r,y-r,b,COPY_PUT);
    x+=dx;
    y+=dy;
    if(y<10 || y>310)
        dy=-dy;
    if(x<10 || x>420)
        dx=-dx;
}
getch();
free(b);
closegraph();
return 0;
}

```

VI. Tóm tắt nội dung bài học

- I. Cây, Hàm băm
- II. Khởi động đồ họa
- III. Các hàm đồ họa
- IV. Xử lý văn bản trên màn hình đồ họa
- V. Hiệu ứng hoạt hình đồ họa

VII. Bài tập

Xem **Bài 19 - Bài tập thực hành** Làm việc với File và Thuật toán nâng cao

Bài 19 - Bài thực hành: MỘT SỐ VẤN ĐỀ MỞ RỘNG

I. Bài tập làm theo yêu cầu

- 1. Cài đặt cây nhị phân tìm kiếm theo ví dụ trong phần I.1.**
- 2. Cài đặt hiệu ứng hoạt hình nêu trong phần V.**

II. Bài tập tự làm

1. Viết chương trình cho phép hiển thị đồng hồ với tương tự (đồng hồ kim) có hai kim giờ và phút. Sử dụng hàm `gettime` để lấy dữ liệu. Theo thời gian các kim chạy hiển thị đúng hiện trạng của đồng hồ máy tính.
2. Viết chương trình mô tả dao động của con lắc đơn.
3. Vẽ đồ thị hàm số $f(x)=3x^3+4x^2-x-1$, trong khoảng $[-4,4]$.

Bài 20 - ÔN TẬP

I. Những vấn đề lý thuyết

Bài 1 - TỔNG QUAN

- I. Giới thiệu
- II. Bài toán và việc giải bài toán trên máy tính
- III. Công cụ lập trình

Bài 2 - NGÔN NGỮ LẬP TRÌNH C/C++

- I. Giới thiệu
- II. Một số khái niệm cơ bản
- III. Cấu trúc một chương trình đơn giản
- IV. Nhập/Xuất dữ liệu

Bài 3 - Bài thực hành: MÔI TRƯỜNG LẬP TRÌNH VÀ CÁC LỆNH VÀO/RA

- I. Làm quen môi trường Dev-C++
- II. Bài tập làm theo yêu cầu
- III. Bài tập tự làm

Bài 4 - ĐIỀU KHIỂN CHỌN

- I. Khối lệnh
- II. Lệnh IF
- III. Lệnh SWITCH

Bài 5 - Bài thực hành: ĐIỀU KHIỂN CHỌN

- I. Bài tập làm theo yêu cầu
- II. Bài tập tự làm

Bài 6 - ĐIỀU KHIỂN LẶP

- I. Lệnh FOR
- II. Lệnh WHILE
- III. Lệnh DO .. WHILE
- IV. Lệnh break và continue

Bài 7 - Bài thực hành: ĐIỀU KHIỂN LẶP

- I. Bài tập làm theo yêu cầu
- II. Bài tập tự làm

Bài 8 - MẢNG VÀ CON TRỎ

- I. Mảng
- II. Con trỏ

Bài 9 - Bài thực hành: MẢNG VÀ CON TRỎ

- I. Bài tập làm theo yêu cầu
- II. Bài tập tự làm

Bài 10 - XÂU KÝ TỰ

- I. Khai báo
- II. Nhập xuất chuỗi
- III. Một số hàm xử lý chuỗi

Bài 11 - Bài thực hành: XÂU KÝ TỰ

- I. Bài tập làm theo yêu cầu
- II. Bài tập tự làm

Bài 12 - HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

- I. Tổ chức chương trình
- II. Hàm do người dùng định nghĩa
- III. Con trỏ hàm
- IV. Đệ qui

Bài 13 - Bài thực hành: HÀM VÀ CẤU TRÚC CHƯƠNG TRÌNH

- I. Bài tập làm theo yêu cầu

- II. Bài tập tự làm
- Bài 14 - CẤU TRÚC DỮ LIỆU DO NGƯỜI DÙNG TỰ ĐỊNH NGHĨA
 - I. Cấu trúc dữ liệu do người dùng tự định nghĩa
 - II. Ngăn xếp
 - III. Hàng đợi
- Bài 15 - Bài thực hành: CẤU TRÚC DỮ LIỆU DO NGƯỜI DÙNG TỰ ĐỊNH NGHĨA
 - I. Bài tập làm theo yêu cầu
 - II. Bài tập tự làm
- Bài 16 - LÀM VIỆC VỚI FILE
 - I. Một số khái niệm
 - II. Các thao tác trên tập tin
 - III. Truy cập tập tin văn bản
 - IV. Truy cập tập tin nhị phân
- Bài 17 - Bài thực hành LÀM VIỆC VỚI FILE
 - I. Bài tập làm theo yêu cầu
 - II. Bài tập tự làm
- Bài 18 - MỘT SỐ VẤN ĐỀ MỞ RỘNG
 - I. Cây, Hàm băm
 - II. Khởi động đồ hoạ
 - III. Các hàm đồ hoạ
 - IV. Xử lý văn bản trên màn hình đồ hoạ
 - V. Hiệu ứng hoạt hình đồ hoạ
- Bài 19 - Bài thực hành: MỘT SỐ VẤN ĐỀ MỞ RỘNG
 - I. Bài tập làm theo yêu cầu
 - II. Bài tập tự làm
- Bài 20 - ÔN TẬP

II. Những vấn đề về thực hành

1. Giải quyết một bài toán trên máy tính bắt đầu như thế nào?
2. Những vấn đề cần chú ý trong lập trình