BỘ GIAO THÔNG VẬN TẢI
TRƯỜNG ĐẠI HỌC HÀNG HẢI
BỘ MÔN: KHOA HỌC MÁY TÍNH
KHOA: CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

KỸ THUẬT LẬP TRÌNH C

TÊN HỌC PHẦN : **KỸ THUẬT LẬP TRÌNH C**
MÃ HỌC PHẦN : **17206**
TRÌNH ĐỘ ĐÀO TẠO : **ĐẠI HỌC CHÍNH QUY**
DÙNG CHO SV NGÀNH : **CÔNG NGHỆ THÔNG TIN**

HẢI PHÒNG - 2008

MỤC LỤC

11.6. **Tên học phần:** Kỹ thuật lập trình (C) **Loại học phần:** 2
Bộ môn phụ trách giảng dạy: Khoa học Máy tính **Khoa phụ trách:** CNTT
Mã học phần: 17206 **Tổng số TC**: 4

TS tiết	Lý thuyết	Thực hành/Xemina	Tự học	Bài tập lớn	Đồ án môn học
75	45	30	0	0	0

Điều kiện tiên quyết:

Sinh viên phải học xong các học phần sau mới được đăng ký học phần này:
Tin đại cương, Toán rời rạc, Đại số, Giải tích 1.

Mục tiêu của học phần:

Cung cấp cho sinh viên kiến thức và rèn luyện kỹ năng lập trình dựa trên ngôn ngữ lập trình C

Nội dung chủ yếu

- Những vấn đề cơ bản về ngôn ngữ lập trình C.
- Cách thức xây dựng một chương trình dựa trên ngôn ngữ lập trình C.
- Các vấn đề về con trỏ, file và đồ họa trong C

Nội dung chi tiết của học phần:

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
Chương 1: Giới thiệu	**2**	**2**	**0**		
1.1. Giới thiệu ngôn ngữ lập trình C.					
1.1.1. Xuất xứ của ngôn ngữ lập trình C.					
1.1.2. Trình biên dịch C và cách sử dụng.					
1.2. Thuật toán và sơ đồ khối					
Chương 2. Các khái niệm cơ bản về ngôn ngữ C	**8**	**4**	**4**		
2.1. Các phần tử cơ bản của ngôn ngữ lập trình C.					
2.2. Cấu trúc chung của chương trình C					
2.3. Các bước cơ bản khi lập chương trình					
2.4. Các kiểu dữ liệu cơ sở					
2.5. Các khai báo trong chương trình C					
2.6. Biểu thức					
2.7. Các hàm toán học					
Chương 3. Các câu lệnh điều khiển của C	**13**	**7**	**5**		**1**
3.1. Lệnh gán giá trị, lệnh gộp					
3.2. Hàm viết dữ liệu ra màn hình					
3.3. Hàm nhập dữ liệu vào từ bàn phím					
3.4. Câu lệnh điều kiện					
3.5. Câu lệnh lựa chọn					
3.6. Câu lệnh lặp for					
3.7. Câu lệnh lặp while					
3.8. Câu lệnh do…while.					
3.9. Câu lệnh break.					
3.10. Lệnh continue					
3.11. Toán tử goto và nhãn (label)					
Chương 4. Hàm	**14**	**8**	**6**		
4.1. Khái niệm về chương trình con					
4.2. Hàm trong C					

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	LT	TH/Xemina	BT	KT
4.3. Chuyển tham số cho hàm					
4.4. Biến toàn cục và biến địa phương					
4.5. Tính đệ quy của hàm					
4.6. Đối dòng lệnh của hàm					
4.7. Một số hàm đặc biệt					
Chương 5. Mảng và kiểu dữ liệu có cấu trúc	**21**	**12**	**8**		**1**
5.1. Dữ liệu kiểu mảng/con trỏ					
5.1.1. Mảng 1 chiều và nhiều chiều					
5.1.2. Con trỏ và địa chỉ					
5.1.3. Liên hệ giữa mảng và con trỏ					
5.1.4. Con trỏ và hàm					
5.2. Dữ liệu kiểu xâu ký tự. Liên hệ giữa con trỏ và xâu ký tự					
5.3. Dữ liệu kiểu bản ghi					
5.4. Một số ví dụ tổng hợp					
Chương 6. File	**10**	**5**	**4**		**1**
6.1. Khái niệm.					
6.2. Cấu trúc và phân loại tệp.					
6.3. Tạo tệp mới để đọc / ghi dữ liệu.					
6.4. Một số hàm xử lý tệp của C.					
6.5. Bài tập áp dụng					
Chương 7. Đồ hoạ trong C	**7**	**4**	**3**		
7.1. Giới thiệu chung					
7.2. Các hàm đặt màu, vẽ điểm, tô màu					
7.3. Các hàm vẽ hình cơ bản					

Nhiệm vụ của sinh viên :

Tham dự các buổi thuyết trình của giáo viên, tự học, tự làm bài tập do giáo viên giao, tham dự các bài kiểm tra định kỳ và cuối kỳ.

Tài liệu tham khảo:

1. Phạm Văn Ất, *Kỹ thuật lập trình C - Cơ sở và nâng cao*, NXB KHKT, 1998.

2. Quách Tuấn Ngọc, *Ngôn ngữ lập trình C*, NXB GD, 1998.

3. Một số website liên quan: http://www.codeproject.com, http://www.cprogramming.com,

Hình thức và tiêu chuẩn đánh giá sinh viên:

Hình thức thi cuối kỳ : Thi vấn đáp trên máy tính, thời gian làm bài 45 phút

Sinh viên phải đảm bảo các điều kiện theo Quy chế của Nhà trường và của Bộ

Thang điểm: Thang điểm chữ A, B, C, D, F

Điểm đánh giá học phần: Z = 0,3X + 0,7Y.

Bài giảng này là tài liệu **chính thức và thống nhất** của Bộ môn Khoa học máy tính, Khoa Công nghệ thông tin và được dùng để giảng dạy cho sinh viên.

Ngày phê duyệt: / /20

Trưởng Bộ môn: ThS. Nguyễn Hữu Tuân (ký và ghi rõ họ tên)

CHƯƠNG 1. GIỚI THIỆU

1.1. Giới thiệu ngôn ngữ lập trình C.

1.1.1. Xuất xứ của ngôn ngữ lập trình C.

Khoảng cuối những năm 1960 đầu 1970 xuất hiện nhu cầu cần có các ngôn ngữ bậc cao để hỗ trợ cho những nhà tin học trong việc xây dựng các phần mềm hệ thống, hệ điều hành. Ngôn ngữ C ra đời từ đó, nó đã được phát triển tại phòng thí nghiệm Bell. Đến năm 1978, giáo trình " Ngôn ngữ lập trình C " do chính các tác giả của ngôn ngữ là Dennish Ritchie và B.W. Kernighan viết, đã được xuất bản và phổ biến rộng rãi.

C là ngôn ngữ lập trình vạn năng. Ngoài việc C được dùng để viết hệ điều hành UNIX, người ta nhanh chóng nhận ra sức mạnh của C trong việc xử lý cho các vấn đề hiện đại của tin học. C không gắn với bất kỳ một hệ điều hành hay máy nào, và mặc dầu nó đã được gọi là " ngôn ngữ lập trình hệ thống" vì nó được dùng cho việc viết hệ điều hành, nó cũng tiện lợi cho cả việc viết các chương trình xử lý số, xử lý văn bản và cơ sở dữ liệu.

1.1.2. Trình biên dịch C và cách sử dụng..

A. Turbo C (TC)

1. Giới thiệu chung TC

Khởi động C cũng như mọi chương trình khác bằng cách nhấp đúp chuột lên biểu tượng của chương trình. Khi chương trình được khởi động sẽ hiện ra giao diện gồm có menu công việc và một khung cửa sổ bên dưới phục vụ cho soạn thảo. Một con trỏ nhấp nháy trong khung cửa sổ và chúng ta bắt đầu nhập nội dung (văn bản) chương trình vào trong khung cửa sổ soạn thảo này. Mục đích của giáo trình này là trang bị những kiến thức cơ bản của lập trình thông qua NNLT C cho các sinh viên mới bắt đầu nên chúng tôi vẫn chọn trình bày giao diện của các trình biên dịch quen thuộc là Turbo C hoặc Borland C. Về các trình biên dịch khác độc giả có thể tự tham khảo trong các tài liệu liên quan.

Để kết thúc làm việc với C (soạn thảo, chạy chương trình ...) và quay về môi trường Windows chúng ta ấn **Alt-X.**

2. Giao diện và cửa sổ soạn thảo của TC

a. Mô tả chung

Khi gọi chạy C trên màn hình sẽ xuất hiện một menu xổ xuống và một cửa sổ soạn thảo. Trên menu gồm có các nhóm chức năng: **File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help**. Để kích hoạt các nhóm chức năng, có thể ấn **Alt+chữ cái** biểu thị cho menu của chức năng đó (là chữ cái có gạch dưới). Ví dụ để mở nhóm chức năng **File** ấn **Alt+F**, sau đó dịch chuyển hộp sáng đến mục cần chọn rồi ấn Enter. Để thuận tiện cho NSD, một số các chức năng hay dùng còn được gắn với một tổ hợp các phím cho phép người dùng có thể chọn nhanh chức năng này mà không cần thông qua việc mở menu như đã mô tả ở trên. Một số tổ hợp phím cụ thể đó sẽ được trình bày vào cuối phần này. Các bộ chương trình dịch hỗ trợ người lập trình một môi trường tích hợp tức ngoài chức năng soạn thảo, nó còn cung cấp nhiều chức năng, tiện ích khác giúp người lập trình vừa có thể soạn thảo văn bản chương trình vừa gọi chạy chương trình vừa gỡ lỗi … Các chức năng liên quan đến soạn thảo phần lớn giống với các bộ soạn thảo khác (như WinWord) do vậy chúng tôi chỉ trình bày tóm tắt mà không trình bày chi tiết ở đây.

b. Các chức năng soạn thảo

Giống hầu hết các bộ soạn thảo văn bản, bộ soạn thảo của Turbo C hoặc Borland C cũng sử dụng các phím sau cho quá trình soạn thảo:

− Dịch chuyển con trỏ: các phím mũi tên cho phép dịch chuyển con trỏ sang trái, phải một kí tự hoặc lên trên, xuống dưới 1 dòng. Để dịch chuyển nhanh có các phím như Home (về đầu

dòng), End (về cuối dòng), PgUp, PgDn (lên, xuống một trang màn hình). Để dịch chuyển xa hơn có thể kết hợp các phím này cùng phím Control (Ctrl, ^) như ^PgUp: về đầu tệp, ^PgDn: về cuối tệp.

− Chèn, xoá, sửa: Phím Insert cho phép chuyển chế độ soạn thảo giữa chèn và đè. Các phím Delete, Backspace cho phép xoá một kí tự tại vị trí con trỏ và trước vị trí con trỏ (xoá lùi).

− Các thao tác với khối dòng: Để đánh dấu khối dòng (thực chất là khối kí tự liền nhau bất kỳ) ta đưa con trỏ đến vị trí đầu ấn Ctrl-KB và Ctrl-KK tại vị trí cuối. Cũng có thể thao tác nhanh hơn bằng cách giữ phím Shift và dùng cácphím dịch chuyển con trỏ quét từ vị trí đầu đến vị trí cuối, khi đó khối kí tự được đánh dấu sẽ chuyển mầu nền. Một khối được đánh dấu có thể dùng để cắt, dán vào một nơi khác trong văn bản hoặc xoá khỏi văn bản. Để thực hiện thao tác cắt dán, đầu tiên phải đưa khối đã đánh dấu vào bộ nhớ đệm bằng nhóm phím Shift-Delete (cắt), sau đó dịch chuyển con trỏ đến vị trí mới cần hiện nội dung vừa cắt và ấn tổ hợp phím Shift-Insert. Một đoạn văn bản được ghi vào bộ nhớ đệm có thể được dán nhiều lần vào nhiều vị trí khác nhau bằngcách lặp lại tổ hợp phím Shift-Insert tại các vị trí khác nhau trong văn bản. Để xoá một khối dòng đã đánh dấu mà không ghi vào bộ nhớ đệm, dùng tổ hợp phím Ctrl-Delete. Khi một nội dung mới ghi vào bộ nhớ đệm thì nó sẽ xoá (ghi đè) nội dung cũ đã có, do vậy cần cân nhắc để sử dụng phím Ctrl-Delete (xoá và không lưu lại nội dung vừa xoá vào bộ đệm) và Shift-Delete (xoá và lưu lại nội dung vừa xoá) một cách phù hợp.

− Tổ hợp phím Ctrl-A rất thuận lợi khi cần đánh dấu nhanh toàn bộ văn bản.

c. Chức năng tìm kiếm và thay thế

Chức năng này dùng để dịch chuyển nhanh con trỏ văn bản đến từ cần tìm. Để thực hiện tìm kiếm bấm Ctrl-QF, tìm kiếm và thay thế bấm Ctrl-QA. Vào từ hoặc nhóm từ cần tìm vào cửa sổ Find, nhóm thay thế (nếu dùng Ctrl-QA) vào cửa sổ Replace và đánh dấu vào các tuỳ chọn trong cửa sổ bên dưới sau đó ấn Enter. Các tuỳ chọn gồm: không phân biệt chữ hoa/thường, tìm từ độc lập hay đứng trong từ khác, tìm trong toàn văn bản hay chỉ trong phần được đánh dấu, chiều tìm đi đến cuối hay ngược về đầu văn bản, thay thế có hỏi lại hay không hỏi lại … Để dịch chuyển con trỏ đến các vùng khác nhau trong một menu hay cửa sổ chứa các tuỳ chọn ta sử dụng phím Tab.

d. Các chức năng liên quan đến tệp

− Ghi tệp lên đĩa: Chọn menu File\Save hoặc phím F2. Nếu tên tệp chưa có (còn mang tên Noname.cpp) máy sẽ yêu cầu cho tên tệp. Phần mở rộng của tên tệp được mặc định là CPP.

− Soạn thảo tệp mới: Chọn menu File\New. Hiện ra cửa sổ soạn thảo trắng và tên file tạm thời lấy là Noname.cpp.

− Soạn thảo tệp cũ: Chọn menu File\Open hoặc ấn phím F3, nhập tên tệp hoặc dịch chuyển con trỏ trong vùng danh sách tệp bên dưới đến tên tệp cần soạn rồi ấn Enter. Cũng có thể áp dụng cách này để soạn tệp mới khi không nhập vào tên tệp cụ thể.

− Ghi tệp đang soạn thảo lên đĩa với tên mới: Chọn menu File\Save As và nhập tên tệp mới vào rồi ấn Enter.

e. Chức năng dịch và chạy chương trình

− Ctrl-F9: Khởi động chức năng dịch và chạy toàn bộ chương trình.

− F4: Chạy chương trình từ đầu đến dòng lệnh hiện tại (đang chứa con trỏ)

− F7: Chạy từng lệnh một của hàm main(), kể cả các lệnh con trong hàm.

− F8: Chạy từng lệnh một của hàm main(). Khi đó mỗi lời gọi hàm được xem là một lệnh (không chạy từng lệnh trong các hàm được gọi). Các chức năng liên quan đến dịch chương trình có thể được chọn thông qua menuCompile (Alt-C).

f. Tóm tắt một số phím nóng hay dùng

− Các phím kích hoạt menu: Alt+chữ cái đại diện cho nhóm menu đó. Ví dụ Alt-F mở menu File để chọn các chức năng cụ thể trong nó như Open (mở file), Save (ghi file lên đĩa), Print (in nội dung văn bản chương trình ra máy in), … Alt-C mở menu Compile để chọn các chức năng dịch chương trình.

− Các phím dịch chuyển con trỏ khi soạn thảo.

− F1: mở cửa sổ trợ giúp. Đây là chức năng quan trọng giúp người lập trình nhớ tên lệnh, cú

pháp và cách sử dụng.
− F2: ghi tệp lên đĩa.
− F3: mở tệp cũ ra sửa chữa hoặc soạn thảo tệp mới.
− F4: chạy chương trình đến vị trí con trỏ.
− F5: Thu hẹp/mở rộng cửa sổ soạn thảo.
− F6: Chuyển đổi giữa các cửa sổ soạn thảo.
− F7: Chạy chương trình theo từng lệnh, kể cả các lệnh trong hàm con.
− F8: Chạy chương trình theo từng lệnh trong hàm chính.
− F9: Dịch và liên kết chương trình. Thường dùng chức năng này để tìm lỗi cú pháp của chương trình nguồn trước khi chạy.
− Alt-F7: Chuyển con trỏ về nơi gây lỗi trước đó.
− Alt-F8: Chuyển con trỏ đến lỗi tiếp theo.
− Ctrl-F9: Chạy chương trình.
− Ctrl-Insert: Lưu khối văn bản được đánh dấu vào bộ nhớ đệm.
− Shift-Insert: Dán khối văn bản trong bộ nhớ đệm vào văn bản tại vị trí con trỏ.
− Shift-Delete: Xoá khối văn bản được đánh dấu, lưu nó vào bộ nhớ đệm.
− Ctrl-Delete: Xoá khối văn bản được đánh dấu (không lưu vào bộ nhớ đệm).
− Alt-F5: Chuyển sang cửa sổ xem kết quả của chương trình vừa chạy xong.
− Alt-X: thoát C về lại Windows.

B. Dev C

C. Visual C++

1.2. Thuật toán và sơ đồ khối

Khái niệm: thuật toán hay giải thuật dùng để chỉ phương pháp hay cách thức (method) để giải quyết vấn đề.

Sơ dồ khối: sử dụng các khối để biểu diễn thuật toán

Bài tập:

1. Vẽ sơ đồ khối giải thuật giải phương trình bậc nhất ax+b=0
2. Vẽ sơ đồ khối giải thuật giải bất phương trình bậc nhất ax+b>0
3. Vẽ sơ đồ khối giải thuật giải phương trình bậc hai $ax^2+bx+c=0$
4. Vẽ sơ đồ khối giải thuật tìm ước số chung lớn nhất của 2 số nguyên dương
5. Vẽ sơ đồ khối giải thuật tìm số nhỏ nhất của 2 số

CHƯƠNG 2. CÁC KHÁI NIỆM CƠ BẢN VỀ NGÔN NGỮ LẬP TRÌNH C

2.1. Các phần tử cơ bản của ngôn ngữ lập trình C.

2.1.1. Tập ký tự dùng trong ngôn ngữ C

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

- 26 chữ cái hoa : A B C .. Z
- 26 chữ cái thường : a b c .. z
- 10 chữ số : 0 1 2 .. 9
- Các ký hiệu toán học : + - * / = ( )
- Ký tự gạch nối : _
- Các ký tự khác : . , : ; [ ] {} ! \ & % # $ ...

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

Chú ý : Khi viết chương trình, ta không được sử dụng bất kỳ ký tự nào khác ngoài các ký tự trên. Ví dụ như khi lập chương trình giải phương trình bậc hai $ax^2+bx+c=0$, ta cần tính biệt thức Delta $\Delta= b^2 - 4ac$, trong ngôn ngữ C không cho phép dùng ký tự Δ, vì vậy ta phải dùng ký hiệu khác để thay thế.

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

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

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

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

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

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

Tên được đặt theo qui tắc sau: Tên là một dãy các ký tự bao gồm chữ cái, số và gạch nối. Ký tự đầu tiên của tên phải là chữ hoặc gạch nối. Tên không được trùng với khoá. Độ dài cực đại của tên theo mặc định là 32 và có thể được đặt lại là một trong các giá trị từ 1 tới 32 nhờ chức năng : Option-Compiler-Source-Identifier length khi dùng TURBO C.

Ví dụ : Các tên đúng : a_1 delta x1 _step GAMA

Các tên sai:

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

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

2.2. Cấu trúc chung của chương trình C

Một chương trình C có thể được đặt trong một hoặc nhiều file văn bản khác nhau. Mỗi file văn bản chứa một số phần nào đó của chương trình. Với những chương trình đơn giản và ngắn thường chỉ cần đặt chúng trên một file.

Một chương trình gồm nhiều hàm, mỗi hàm phụ trách một công việc khác nhau của chương trình. Đặc biệt trong các hàm này có một hàm duy nhất có tên hàm là main(). Khi chạy chương trình, các câu lệnh trong hàm main() sẽ được thực hiện đầu tiên. Trong hàm main() có thể có các câu lệnh gọi đến các hàm khác khi cần thiết, và các hàm này khi chạy lại có thể gọi đến các hàm khác nữa đã được viết trong chương trình (trừ việc gọi quay lại hàm main()). Sau khi chạy đến lệnh cuối cùng của hàm main() chương trình sẽ kết thúc.

Cụ thể, thông thường một chương trình gồm có các nội dung sau:

− Phần khai báo các tệp nguyên mẫu: khai báo tên các tệp chứa những thành phần có sẵn (như các hằng chuẩn, kiểu chuẩn và các hàm chuẩn) mà NSD sẽ dùng trong chương trình.

− Phần khai báo các kiểu dữ liệu, các biến, hằng ... do NSD định nghĩa và được dùng chung trong toàn bộ chương trình.

− Danh sách các hàm của chương trình (do NSD viết, bao gồm cả hàm main()). Cấu trúc chi tiết của mỗi hàm sẽ được đề cập đến trong chương 4.

Dưới đây là một đoạn chương trình đơn giản chỉ gồm 1 hàm chính là hàm main().

Nội dung của chương trình dùng in ra màn hình dòng chữ: *Chào các bạn, bây giờ là 2giờ.*

```
#include <stdio.h>// khai báo tệp nguyên mẫu để được sử dụng hàm printf, scanf
void main()
```

```
{
int h = 2; // Khai báo và khởi tạo biến h = 2
printf( “Chào các bạn, bây giờ là  %d  giờ”,h) ;// in ra màn hình
}
```

Dòng đầu tiên của chương trình là khai báo tệp nguyên mẫu stdio.h. Đây là khai báo bắt buộc vì trong chương trình có sử dụng hàm chuẩn printf() (in ra màn hình), hàm này được khai báo và định nghĩa sẵn trong stdio.h.

Không riêng hàm main(), mọi hàm khác đều phải bắt đầu tập hợp các câu lệnh của mình bởi dấu { và kết thúc bởi dấu }. Tập các lệnh bất kỳ bên trong cặp dấu này được gọi là khối lệnh. Khối lệnh là một cú pháp cần thiết trong các câu lệnh có cấu trúc như ta sẽ thấy trong các chương tiếp theo.

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

- Các #include
- Các #define
- Khai báo các đối tượng dữ liệu ngoài (biến, mảng, cấu trúc vv..).
- Khai báo nguyên mẫu các hàm.
- Hàm main().
- Định nghĩa các hàm (hàm main có thể đặt sau hoặc xen vào giữa các hàm khác).

2.3. Các bước cơ bản khi lập chương trình

2.3.1. Qui trình viết và thực hiện chương trình

Trước khi viết và chạy một chương trình thông thường chúng ta cần:

1. Xác định yêu cầu của chương trình. Nghĩa là xác định dữ liệu đầu vào (input) cung cấp cho chương trình và tập các dữ liệu cần đạt được tức đầu ra (output).Các tập hợp dữ liệu này ngoài các tên gọi còn cần xác định kiểu của nó.Ví dụ để giải một phương trình bậc 2 dạng: $ax^2 + bx + c = 0$, cần báo cho chương trình biết dữ liệu đầu vào là a, b, c và đầu ra là nghiệm x1 và x2 của phương trình. Kiểu của a, b, c, x1, x2 là các số thực.
2. Xác định thuật toán giải.
3. Cụ thể hoá các khai báo kiểu và thuật toán thành dãy các lệnh, tức viết thành chương trình thông thường là trên giấy, sau đó bắt đầu soạn thảo vào trong máy. Quá trình này được gọi là soạn thảo chương trình nguồn.
4. Dịch chương trình nguồn để tìm và sửa các lỗi gọi là lỗi cú pháp.
5. Chạy chương trình, kiểm tra kết quả in ra trên màn hình. Nếu sai, sửa lại chương trình, dịch và chạy lại để kiểm tra. Quá trình này được thực hiện lặp đi lặp lại cho đến khi chương trình chạy tốt theo yêu cầu đề ra của NSD.

2.3.2. Soạn thảo tệp chương trình nguồn

Soạn thảo chương trình nguồn là một công việc đơn giản: gõ nội dung của chương trình (đã viết ra giấy) vào trong máy và lưu lại nó lên đĩa. Thông thường khi đã lưu lại chương trình lên đĩa lần sau sẽ không cần phải gõ lại. Có thể soạn chương trình nguồn trên các bộ soạn thảo (editor) khác nhưng phải chạy trong môi trường tích hợp C++ (Borland C, Turbo C). Mục đích của soạn thảo là tạo ra một văn bản chương trình và đưa vào bộ nhớ của máy. Văn bản chương trình cần được trình bày sáng sủa, rõ ràng. Các câu lệnh cần gióng thẳng cột theo cấu trúc của lệnh (các lệnh chứa trong một lệnh cấu trúc được trình bày thụt vào trong so với điểm bắt đầu của lệnh). Các chú thích nên ghi ngắn gọn, rõ nghĩa và phù hợp.

2.3.3. Dịch chương trình

Sau khi đã soạn thảo xong chương trình nguồn, bước tiếp theo thường là dịch (ấn tổ hợp phím

Alt-F9) để tìm và sửa các lỗi gọi là lỗi cú pháp. Trong khi dịch C++ sẽ đặt con trỏ vào nơi gây lỗi (viết sai cú pháp) trong văn bản. Sau khi sửa xong một lỗi NSD có thể dùng Alt-F8 để chuyển con trỏ đến lỗi tiếp theo hoặc dịch lại. Để chuyển con trỏ về ngược lại lỗi trước đó có thể dùng Alt-F7. Quá trình sửa lỗi − dịch được lặp lại cho đến khi văn bản đã được sửa hết lỗi cú pháp.

Sản phẩm sau khi dịch là một tệp mới gọi là chương trình đích có đuôi EXE tức là tệp mã máy để thực hiện.Tệp này có thể lưu tạm thời trong bộ nhớ phục vụ cho quá trình chạy chương trình hoặc lưu lại trên đĩa tuỳ theo tuỳ chọn khi dịch của NSD. Trong và sau khi dịch, C++ sẽ hiện một cửa sổ chứa thông báo về các lỗi (nếu có), hoặc thông báo chương trình đã được dịch thành công (không còn lỗi). Các lỗi này được gọi là lỗi cú pháp.

Để dịch chương trình ta chọn menu \Compile\Compile hoặc \Compile\Make hoặc nhanh chóng hơn bằng cách ấn tổ hợp phím Alt-F6.

2.3.4. Chạy chương trình

Ấn Ctrl-F9 để chạy chương trình, nếu chương trình chưa dịch sang mã máy, máy sẽ tự động dịch lại trước khi chạy. Kết quả của chương trình sẽ hiện ra trong một cửa sổ kết quả để NSD kiểm tra. Nếu kết quả chưa được như mong muốn, quay lại văn bản để sửa và lại chạy lại chương trình. Quá trình này được lặp lại cho đến khi chương trình chạy đúng như yêu cầu đã đề ra. Khi chương trình chạy, cửa sổ kết quả sẽ hiện ra tạm thời che khuất cửa sổ soạn thảo. Sau khi kết thúc chạy chương trình cửa sổ soạn thảo sẽ tự động hiện ra trở lại và che khuất cửa sổ kết quả. Để xem lại kết quả đã hiện ấn Alt-F5 (hoặc thêm lệnh getch() vào cuối hàm main()). Sau khi xem xong để quay lại cửa sổ soạn thảo ấn phím bất kỳ.

2.4. Các kiểu dữ liệu cơ sở

Trong C sử dụng các các kiểu dữ liệu cơ sở sau :

2.4.1. Kiểu ký tự (char):

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

Ký tự	Mã ASCII
0	048
1	049
2	050
A	065
B	066
a	097
b	098

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

Kiểu	**Phạm vi biểu diễn**	**Số ký tự**	**Kích thước**
char (signed char)	-128 đến 127	256	1 byte
unsigned char	0 đến 255	256	1 byte

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

```
char  ch1;
```

```
unsigned char ch2;
.....
ch1=200; ch2=200;
```

Khi đó thực chất :

```
ch1=-56;
ch2=200;
```

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

Phân loại ký tự :

Có thể chia 256 ký tự làm ba nhóm :

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

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

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

2.4.2. Kiểu số nguyên :

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

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

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

2.4.3. Kiểu dấu phảy động (số thực):

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

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

Giải thích:

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

Chú ý: Trong C không có kiểu logic Boolean (thể hiện giá trị True, False). C sử dụng kiểu số nguyên để xây dựng kiểu logic, 0 ứng với False, ≠ 0 ứng với trị True. Ví dụ: biểu thức 6>8 nhận giá trị 0, 6>3 nhận giá trị 1.

2.4.4. Định nghĩa kiểu bằng typedef :

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

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

typedef int nguyen;

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

nguyen x, y;

2.4.5. Các phép toán số học, quan hệ và logic

Các phép toán số học.

Các phép toán hai ngôi số học là

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

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

Ví dụ :11/3=3

11%3=2

-(2+6)=-8

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

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

Các phép toán quan hệ và logic :

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

Các phép toán quan hệ là :

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

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

Các phép toán logic :

Trong C sử dụng ba phép toán logic :

Phép phủ định một ngôi !

a	!a
1	0
0	1

Phép và (AND) &&

Phép hoặc (OR) ||

A	B	A&&B	A || B
1	1	1	1
1	0	0	1
0	1	0	1

0	0	0	0

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

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

Phép toán tăng giảm :

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

Ví dụ: n=5

++n Cho ta n=6

--n Cho ta n=4

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

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

Ví dụ: n=5

x=++n Cho ta x=6 và n=6

x=n++ Cho ta x=5 và n=6

Thứ tự ưu tiên các phép toán :

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

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

TT	Phép toán	Trình tự kết hợp
1	() [] ->	Trái qua phải
2	! ~ & * - ++ -- (type) sizeof	Phải qua trái
3	* (phép nhân) / %	Trái qua phải
4	+ -	Trái qua phải
5	<< >>	Trái qua phải
6	< <= > >=	Trái qua phải
7	== !=	Trái qua phải
8	&	Trái qua phải
9	^	Trái qua phải
10	|	Trái qua phải
11	&&	Trái qua phải
12	||	Trái qua phải

13	?:	Phải qua trái
14	= += -= *= /= %= <<= >>= &= ^= |=	Phải qua trái
15	,	Trái qua phải

Chú thích: Các phép toán tên một dòng có cùng thứ tự ưu tiên, các phép toán ở hàng trên có số ưu tiên cao hơn các số ở hàng dưới. Đối với các phép toán cùng mức ưu tiên thì trình tự tính toán có thể từ trái qua phải hay ngược lại được chỉ ra trong cột *trình tự kết hợp*.

Ví dụ: *--px=*(--px) (Phải qua trái)

8/4*6=(8/4)*6 (Trái qua phải)

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

Các phép toán lạ :

Dòng 1

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

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

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

Dòng 2

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

& Phép toán lấy địa chỉ, ví dụ : &x

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

Dòng 15

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

2.5. Các khai báo trong chương trình C

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

Tên hằng: Nguyên tắc đặt tên hằng ta đã xem xét trong mục đặt tên ở phần trước.

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

Để khai báo hằng ta dùng các câu khai báo sau:

#define tên_hằng giá_trị_hằng

hoặc:

const tên_hằng = giá_trị_hằng ;

Ví dụ:

```
#define  sosv 50
#define  MAX 100
const  sosv = 50 ;
```

Lúc này, tất cả các tên MAX trong chương trình xuất hiện sau này đều được thay bằng 100. Vì vậy, ta thường gọi MAX là tên hằng, nó biểu diễn số 100.

Một ví dụ khác : #define pi 3.141593

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

Các loại hằng :

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

Ví dụ : #define number1 -50 Định nghiã hằng int number1 có giá trị là -50

#define sodem 2732 Định nghiã hằng int sodem có giá trị là 2732

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

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

Hằng long được viết theo cách : 1234L hoặc 1234l (thêm L hoặc l vào đuôi)

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

Ví dụ : #define sl 8865056L Định nghiã hằng long sl có giá trị là 8865056

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

Hằng int hệ 8: Hằng int hệ 8 được viết theo cách 0c1c2c3....ở đây ci là một số nguyên dương trong khoảng từ 1 đến 7. Hằng int hệ 8 luôn luôn nhận giá trị dương.

Ví dụ: #define h8 0345 Định nghiã hằng int hệ 8 có giá trị là

3*8*8+4*8+5=229

Hằng int hệ 16: Trong hệ này ta sử dụng 16 ký tự : 0,1..,9,A,B,C,D,E,F.

Cách viết	Giá trị
a hoặc A	10
b hoặc B	11
c hoặc C	12
d hoặc D	13
e hoặc E	14
f hoặc F	15

Hằng số hệ 16 có dạng 0xc1c2c3... hặc 0Xc1c2c3... ở đây ci là một số trong hệ 16.

Ví dụ :

```
#define h16 0xa5
#define h16 0xA5
#define h16 0Xa5
#define h16 0XA5
```

Cho ta các hắng số h16 trong hệ 16 có giá trị như nhau. Giá trị của chúng trong hệ 10 là: 10*16+5=165.

Hằng ký tự: Hằng ký tự là một ký tự riêng biệt được viết trong hai dấu nháy đơn, ví dụ 'a'. Giá trị của 'a' chính là mã ASCII của chữ a. Như vậy giá trị của 'a' là 97. Hằng ký tự có thể tham gia vào các phép toán như mọi số nguyên khác. Ví dụ : '9'-'0'=57-48=9

Ví dụ : #define kt 'a' Định nghiã hằng ký tự kt có giá trị là 97

Hằng ký tự còn có thể được viết theo cách sau: ' \c1c2c3' .Trong đó c1c2c3 là một số hệ 8 mà giá trị của nó bằng mã ASCII của ký tự cần biểu diễn.

Ví dụ : chữ a có mã hệ 10 là 97, đổi ra hệ 8 là 0141. Vậy hằng ký tự 'a' có thể viết dưới dạng '\141'. Đối với một vài hằng ký tự đặc biệt ta cần sử dụng cách viết sau (thêm dấu \ ) :

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

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

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

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

Ví dụ : #define xau1 "Ha noi"

#define xau2 "My name is Giang"

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

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

2.5.2. Biến. Là đại lượng mà giá trị có thể thay đổi được trong chương trình. Mỗi biến cần phải được khai báo trước khi đưa vào sử dụng, giá trị của biến có thể thay đổi được trong chương trình. Việc khai báo biến được thực hiện theo mẫu sau:

Kiểu_dữ_liệu_của_biến tên biến ;

Ví dụ : int a,b,c; Khai báo ba biến int là a,b,c

long dai,mn; Khai báo hai biến long là dai và mn

char kt1,kt2;	Khai báo hai biến ký tự là kt1 và kt2
float x,y	Khai báo hai biến float là x và y
double canh1, canh2;	Khai báo hai biến double là canh1 và canh2

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

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

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

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

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

Ví dụ :
```
int a, b=20, c, d=40;
        float e=-55.2, x=27.23, y, z, t=18.98;
```

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

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

2.5.3. Chuyển đổi kiểu giá trị:

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

+ Khi gán biểu thức gồm các toán hạng khác kiểu.

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

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

Ví dụ :
```
(float) (a+b)
```

Chuyển đổi kiểu trong biểu thức :

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

Ví dụ: 1.5*(11/3)=4.5

1.5*11/3=5.5

(11/3)*1.5=4.5

Chuyển đổi kiểu thông qua phép gán :

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

Ví dụ :int n;

n=15.6 giá trị của n là 15

Đổi kiểu dạng (type)biểu thức :

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

Ví dụ : Phép toán : (int)a

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

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

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

Chú ý:Muốn có giá trị chính xác trong phép chia hai số nguyên cần dùng phép ép kiểu : (float)a/b

Để đổi giá trị thực r sang nguyên, ta dùng: (int)(r+0.5)

Chú ý thứ tự ưu tiên:

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

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

2.6. Biểu thức

Biểu thức là dãy kí hiệu kết hợp giữa các toán hạng, phép toán và cặp dấu () theo một qui tắc nhất định. Các toán hạng là hằng, biến, hàm. Biểu thức cung cấp một cách thức để tính giá trị mới dựa trên các toán hạng và toán tử trong biểu thức. Như vậy hằng, biến, phần tử mảng và hàm cũng được xem là biểu thức.

Ví dụ:

(x + y) * 2 - 4 ;
3 - x + sqrt(y) ;
(-b + sqrt(delta)) / (2*a) ;
Trong C, ta có hai khái niệm về biểu thức :

Biểu thức gán.

Biểu thức điều kiện.

Biểu thức được phân loại theo kiểu giá trị: nguyên và thực. Trong các mệnh đề logic, biểu thức được phân thành đúng (giá trị khác 0) và sai (giá trị bằng 0, chúng ta thường quy ước là 1).

Biểu thức thường được dùng trong:

+ Vế phải của câu lệnh gán.

+ Làm tham số thực sự của hàm.

+ Làm chỉ số.

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

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

2.7. Các hàm toán học

2.7.1. Các hàm số học

• abs(x), labs(x), fabs(x) : trả lại giá trị tuyệt đối của một số nguyên, số nguyên dài và số thực.
• pow(x, y) : hàm mũ, trả lại giá trị x lũy thừa y (x^y).
• exp(x) : hàm mũ, trả lại giá trị e mũ x (e^x).
• log(x), log10(x) : trả lại lôgarit cơ số e và lôgarit thập phân của x (lnx, logx) .
• sqrt(x) : trả lại căn bậc 2 của x.
• atof(s_number) : trả lại số thực ứng với số viết dưới dạng xâu kí tự s_number.

2.7.2. Các hàm lượng giác

• sin(x), cos(x), tan(x) : trả lại các giá trị sinx, cosx, tgx.

Bài tập:

1. Viết chương trình sử dụng các hàm toán học để tính toán giá trị một số biểu thức
2. Tìm hiểu cách thức dịch một chương trình bằng một trong các chương trình dịch TC, C-Free, Dev C, …

CHƯƠNG 3. CÁC CÂU LỆNH CƠ BẢN

3.1. Lệnh gán giá trị, lệnh gộp

Lệnh gán giá trị:

Biểu thức gán (lệnh gán) là biểu thức có dạng: v = e

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

Biểu thức gán có thể sử dụng trong các phép toán và các câu lệnh như các biểu thức khác. Ví dụ như khi ta viết a=b=5;

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

Hoàn toàn tương tự như: a=b=c=d=6; gán 6 cho cả a, b, c và d

Ví dụ: z=(y=2)*(x=6); { ở đây * là phép toán nhân } gán 2 cho y, 6 cho x và nhân hai biểu thức lại cho ta z=12.

Lệnh gộp (khối lệnh):

Một **câu lệnh** trong C được thiết lập từ các từ khoá và các biểu thức … và luôn luôn được kết thúc bằng dấu chấm phẩy. Các ví dụ vào/ra hoặc các phép gán tạo thành những câu lệnh đơn giản như:

```
x = 3 + x ;
y = (x = sqrt(x)) + 1 ;
printf(“x = %4d, y=%4.2f”, x, y );
```

Các câu lệnh được phép viết trên cùng một hoặc nhiều dòng. Một số câu lệnh được gọi là lệnh có cấu trúc, tức bên trong nó lại chứa dãy lệnh khác. Dãy lệnh này phải được bao giữa cặp dấu ngoặc {} và được gọi là *khối lệnh*. Ví dụ tất cả các lệnh trong một hàm (như hàm main()) luôn luôn là một khối lệnh. Một đặc điểm của khối lệnh là các biến được khai báo trong khối lệnh nào thì chỉ có tác dụng trong khối lệnh đó.

Một dãy các câu lệnh được bao bởi các dấu { } gọi là một khối lệnh. Ví dụ :

```
{
        a=2;
        b=3;
        printf("\n%6d%6d",a,b);
}
```

TURBO C xem khối lệnh cũng như một câu lệnh riêng lẻ. Nói cách khác, chỗ nào viết được một câu lệnh thì ở đó cũng có quyền đặt một khối lệnh.

Khai báo ở đầu khối lệnh :

Các khai báo biến và mảng chẳng những có thể đặt ở đầu của một hàm mà còn có thể viết ở đầu khối lệnh :

```
{
        int a, b;
```

```
        float x, y, z;
        a=b=3;
        x=5.5; y=a*x;
        z=b*x;
        printf("\n y= %8.2f\n z=%8.2f",y,z);
}
```

Sự lồng nhau của các khối lệnh và phạm vi hoạt động của các biến và mảng :

Bên trong một khối lệnh lại có thể viết lồng khối lệnh khác. Sự lồng nhau theo cách như vậy là không hạn chế. Khi máy bắt đầu làm việc với một khối lệnh thì các biến và mảng khai báo bên trong nó mới được hình thành và được hình thành và được cấp phát bộ nhớ. Các biến này chỉ tồn tại trong thời gian máy làm việc bên trong khối lệnh và chúng lập tức biến mất ngay sau khi máy ra khỏi khối lệnh. Vậy: Giá trị của một biến hay một mảng khai báo bên trong một khối lệnh không thể đưa ra sử dụng ở bất kỳ chỗ nào bên ngoài khối lệnh đó.

Ở bất kỳ chỗ nào bên ngoài một khối lệnh ta không thể can thiệp đến các biến và các mảng được khai báo bên trong khối lệnh. Nếu bên trong một khối ta dùng một biến hay một mảng có tên là a thì điều này không làm thay đổi giá trị của một biến khác cũng có tên là a (nếu có) được dùng ở đâu đó bên ngoài khối lệnh này. Nếu có một biến đã được khai báo ở ngoài một khối lệnh và không trùng tên với các biến khai báo bên trong khối lệnh này thì biến đó cũng có thể sử dụng cả bên trong cũng như bên ngoài khối lệnh.

Ví dụ : Xét đoạn chương trình sau :

```
{
        int a=5,b=2;
        {
                int a=4;
                b=a+b;
                printf("\n a trong =%3d  b=%3d",a,b);
        }
        printf("\n a ngoai =%3d  b=%3d",a,b);
}
```

Khi đó đoạn chương trình sẽ in kết quả như sau :

a trong =4 b=6

a ngoài =5 b=6

Do tính chất biến a trong và ngoài khối lệnh.

3.2. Hàm viết dữ liệu ra màn hình

3.2.1. Hàm putchar ():

Để đưa một ký tự ra thiết bị ra chuẩn, nói chung là màn hình, ta sử dụng hàm putchar()

Cách dùng: Dùng câu lệnh sau: putchar(ch);

Công dụng: Đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Ký tự sẽ được hiển thị với màu trắng.

Ví dụ:
```
int c;
c = getchar();
putchar(c);
```

3.2.2. Hàm putch():

Cách dùng: Dùng câu lệnh sau: putch(ch);

Công dụng: Đưa ký tự ch lên màn hình tại vị trí hiện tại của con trỏ. Ký tự sẽ được hiển thị theo màu xác định trong hàm textcolor. Hàm cũng trả về ký tự được hiển thị.

3.2.3. Đưa kết quả lên màn hình - hàm printf :

Cách dùng: prinf(điều khiển, đối số 1, đối số 2, ...);

Hàm printf chuyển, tạo khuôn dạng và in các đối của nó ra thiết bị ra chuẩn dưới sự điều khiển của xâu *điều khiển*. Xâu *điều khiển* chứa hai kiểu đối tượng : các ký tự thông thường, chúng sẽ được đưa ra trực tiếp thiết bị ra, và các đặc tả chuyển dạng, mỗi đặc tả sẽ tạo ra việc đổi dạng và in đối tiếp sau của printf.

Chuỗi *điều khiển* có thể có các ký tự điều khiển:

\n sang dòng mới; \f sang trang mới; \b lùi lại một bước; \t dấu tab

Dạng tổng quát của đặc tả :

%[-][n][.m] **ký_tự_chuyển_dạng**

Mỗi đặc tả chuyển dạng đều được đưa vào bằng ký tự % và kết thúc bởi một ***ký_tự_chuyển_dạng***. Giữa % và ***ký_tự_chuyển_dạng*** có thể có:

Dấu trừ: Khi không có dấu trừ thì kết quả ra được dồn về bên phải nếu độ dài thực tế của kết quả ra nhỏ hơn độ rộng tối thiểu n dành cho nó. Các vị trí dư thừa sẽ được lấp đầy bằng các khoảng trống. Riêng đối với các trường số, nếu dãy số n bắt đầu bằng số 0 thì các vị trí dư thừa bên trái sẽ được lấp đầy bằng các số 0.

Khi có dấu trừ thì kết quả được dồn về bên trái và các vị trí dư thừa về bên phải (nếu có) luôn được lấp đầy bằng các khoảng trống.

n : Khi **n** lớn hơn độ dài thực tế của kết quả ra thì các vị trí dư thừa sẽ được lấp đầy bởi các khoảng trống hoặc số 0 và nội dung của kết quả ra sẽ được đẩy về bên phải hoặc bên trái. Khi không có **n** hoặc **n** nhỏ hơn hay bằng độ dài thực tế của kết quả ra thì độ rộng trên

thiết bị ra dành cho kết quả sẽ bằng chính độ dài của nó. Tại vị trí của **n** ta có thể đặt dấu *, khi đó **n** được xác định bởi giá trị nguyên của đối tương ứng.

Ví dụ :

Kết quả ra	**n**	**Dấu -**	**Kết quả đưa ra**
-2503	8	có	-2503
-2503	08	có	-2503
-2503	8	không	-2503
-2503	08	không	000-2503
"abcdef"	8	không	abcdef
"abcdef"	08	có	abcdef
"abcdef"	08	không	abcdef

m: Tham số **m** chỉ được sử dụng khi đối tương ứng là một xâu ký tự hoặc một giá trị kiểu float hay double. Trong trường hợp đối tương ứng có giá trị kiểu float hay double thì **m** là độ chính xác của trường ra. Nói một cách cụ thể hơn giá trị in ra sẽ có pp chữ số sau số thập phân. Khi vắng mặt pp thì độ chính xác sẽ được xem là 6. Khi đối là xâu ký tự: Nếu m nhỏ hơn độ dài của xâu thì chỉ pp ký tự đầu tiên của xâu được in ra. Nếu không có n hoặc nếu m lớn hơn hay bằng độ dài của xâu thì cả xâu ký tự sẽ được in ra.

Ví dụ :

Kết quả ra	**n**	**m**	**Dấu -**	**Kết quả đưa ra**	**Độ dài trường ra**
-435.645	10	2	có	-435.65	7
-435.645	10	0	có	-436	4
-435.645	8	vắng	có	-435.645000	11
"alphabeta"	8	3	vắng	alp	3
"alphabeta"	vắng	vắng	vắng	alphabeta	9
"alpha"	8	6	có	alpha	5

Các ký tự chuyển dạng và ý nghĩa của nó :

Ký tự chuyển dạng là một hoặc một dãy ký hiệu xác định quy tắc chuyển dạng và dạng in ra của đối tương ứng. Như vậy sẽ có tình trạng cùng một số sẽ được in ra theo các dạng khác nhau. Cần phải sử dụng các ký tự chuyển dạng theo đúng qui tắc định sẵn. Bảng sau cho các thông tin về các ký tự chuyển dạng.

Ký tự chuyển dạng	**Ý nghĩa**
D	Đối được chuyển sang số nguyên hệ thập phân
O	Đối được chuyển sang hệ tám không dấu (không có số 0 đứng trước)

x	Đối được chuyển sang hệ mười sáu không dấu (không có 0x đứng trước)
u	Đối được chuyển sang hệ thập phân không dấu
c	Đối được coi là một ký tự riêng biệt
s	Đối là xâu ký tự, các ký tự trong xâu được in cho tới khi gặp ký tự không hoặc cho tới khi đủ số lượng ký tự được xác định bởi các đặc tả về độ chính xác m.
e	Đối được xem là float hoặc double và được chuyển sang dạng thập phân có dạng [-]m.n..nE[+ hoặc -] với độ dài của xâu chứa n là pp.
f	Đối được xem là float hoặc double và được chuyển sang dạng thập phân có dạng [-]m..m.n..n với độ dài của xâu chứa n là pp. Độ chính xác mặc định là 6. Lưu ý rằng độ chính xác không xác định ra số các chữ số có nghĩa phải in theo khuôn dạng f.
g	Dùng %e hoặc %f, tuỳ theo loại nào ngắn hơn, không in các số 0 vô nghĩa.

Chú ý: Mọi dãy ký tự không bắt đầu bằng % hoặc không kết thúc bằng ký tự chuyển dạng đều được xem là ký tự hiển thị.

Để hiển thị các ký tự đặc biệt :

Cách viết	Hiển thị
\'	'
\"	"
\\	\

Các ví dụ :

1	printf("\" Nang suat tang : %d % \" \n\\d"",30,-50);	"Nang suat tang ; 30 %" \d=-50

2	n=8;
	float x=25.5, y=-47.335
	printf("\n%f\n%*.2f",x,n,y);
	Lệnh này tương đương với
	printf("\n%f\n%8.2f",x,n,y);
	Vì n=8 tương ứng với vị trí *
		25.500000
		-47.34

3.3. Hàm nhập dữ liệu vào từ bàn phím

3.3.1. Hàm getchar (): Cơ chế vào đơn giản nhất là đọc từng ký tự từ thiết bị vào chuẩn, nói chung là bàn phím và màn hình của người sử dụng, bằng hàm getchar().

Cách dùng: Dùng câu lệnh sau: biến = getchar();

Công dụng: Nhận một ký tự vào từ bàn phím và không đưa ra màn hình. Hàm sẽ trả về ký tự nhận được và lưu vào biến.

Ví dụ: int c;

c = getchar();

3.3.2. Hàm getch(): Hàm nhận một ký tự từ bộ đệm bàn phím, không cho hiện lên màn hình.

Cách dùng: Dùng câu lệnh sau: getch();

Công dụng : Nếu có sẵn ký tự trong bộ đệm bàn phím thì hàm sẽ nhận một ký tự trong đó.

Nếu bộ đệm rỗng, máy sẽ tạm dừng. Khi gõ một ký tự thì hàm nhận ngay ký tự đó (không cần bấm thêm phím Enter như trong các hàm nhập khác). Ký tự vừa gõ không hiện lên màn hình.

Nếu dùng: biến=getch(); Thì biến sẽ chứa ký tự đọc vào.

Ví dụ: c = getch();

3.3.3. Vào số liệu từ bàn phím - hàm scanf :

Hàm scanf là hàm đọc thông tin từ thiết bị vào chuẩn (bàn phím), chuyển dịch chúng (thành số nguyên, số thực, ký tự vv..) rồi lưu trữ nó vào bộ nhớ theo các địa chỉ xác định.

Cách dùng: scanf(điều khiển,đối 1, đối 2, ...);

Xâu *điều khiển* chứa các đặc tả chuyển dạng, mỗi đặc tả sẽ tạo ra việc đổi dạng biến tiếp sau của scanf.

Đặc tả có thể viết một cách tổng quát như sau :

%[*][d...d]ký tự chuyển dạng

Việc có mặt của dấu * nói lên rằng trường vào vẫn được dò đọc bình thường, nhưng giá trị của nó bị bỏ qua (không được lưu vào bộ nhớ). Như vậy đặc tả chứa dấu * sẽ không có đối tương ứng.

d...d: là một dãy số xác định chiều dài cực đại của trường vào, ý nghĩa của nó được giải thích như sau:

Nếu tham số d...d vắng mặt hoặc nếu giá trị của nó lớn hơn hay bằng độ dài của trường vào tương ứng thì toàn bộ trường vào sẽ được đọc, nội dung của nó được dịch và được gán cho địa chỉ tương ứng (nếu không có dấu *).

Nếu giá trị của d...d nhỏ hơn độ dài của trường vào thì chỉ phần đầu của trường có kích cỡ bằng d...d được đọc và gán cho địa chỉ của biến tương ứng. Phần còn lại của trường sẽ được xem xét bởi các đặc tả và đối tương ứng tiếp theo.

Ví dụ :int a;

float x, y;

char ch[6],ct[6] ;//khai báo xâu ký tự

scanf("%f%5f%3d%3s%s",&x&y&a&ch&ct0;

Với dòng vào : 54.32e-1 25 12452348a

Kết quả là lệnh scanf sẽ gán

5.432 cho x

25.0 cho y

124 cho a

xâu "523" và dấu kết thúc \0 cho ch

xâu "48a" và dấu kết thúc \0 cho ct

Ký tự chuyển dạng: Ký tự chuyển dạng xác định cách thức dò đọc các ký tự trên dòng vào cũng như cách chuyển dịch thông tin đọc được trước khi gán nó cho các địa chỉ tương ứng.

Cách dò đọc thứ nhất là đọc theo trường vào, khi đó các khoảng trắng bị bỏ qua. Cách này áp dụng cho hầu hết các trường hợp.

Cách dò đọc thứ hai là đọc theo ký tự, khi đó các khoảng trắng cũng được xem xét bình đẳng như các ký tự khác. Phương pháp này chỉ xảy ra khi ta sử dụng một trong ba ký tự chuyển dạng sau : C, [dãy ký tự], [^ dãy ký tự]

Các ký tự chuyển dạng và ý nghĩa của nó :

c	Vào một ký tự, đối tương ứng là con trỏ ký tự. Có xét ký tự khoảng trắng
d	Vào một giá trị kiểu int, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên
ld	Vào một giá trị kiểu long, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên
o	Vào một giá trị kiểu int hệ 8, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 8
lo	Vào một giá trị kiểu long hệ 8, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 8
x	Vào một giá trị kiểu int hệ 16, đối tương ứng là con trỏ kiểu int. Trường phải vào là số nguyên hệ 16
lx	Vào một giá trị kiểu long hệ 16, đối tương ứng là con trỏ kiểu long. Trường phải vào là số nguyên hệ 16
f hay e	Vào một giá trị kiểu float, đối tương ứng là con trỏ float, trường vào phải là số dấu phảy động
lf hay le	Vào một giá trị kiểu double, đối tương ứng là con trỏ double, trường vào phải là số dấu phảy động
s	Vào một giá trị kiểu double, đối tương ứng là con trỏ kiểu char, trường vào phải

	là dãy ký tự bất kỳ không chứa các dấu cách và các dấu xuống dòng

[Dãy ký tự], [^Dãy ký tự] Các ký tự trên dòng vào sẽ lần lượt được đọc cho đến khi nào gặp một ký tự không thuộc tập các ký tự đặt trong[]. Đối tương ứng là con trỏ kiểu char. Trường vào là dãy ký tự bất kỳ (khoảng trắng được xem như một ký tự).

Ví dụ :

```
int a,b;
char ch[10], ck[10];
scanf("%d%[0123456789]%[^0123456789]%3d",&a,ch,ck,&b);
```

Với dòng vào: 35 13145 xyz 584235

Sẽ gán: 35 cho a

xâu "13145" cho ch

xâu "xyz' cho ck

584 cho b

Chú ý: Xét đoạn chương trình dùng để nhập (từ bàn phím) ba giá trị nguyên rồi gán cho ba biến a,b,c như sau: int a,b,c;

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

Để vào số liệu ta có thể thao tác theo nhiều cách khác nhau:

Cách 1: Đưa ba số vào cùng một dòng, các số phân cách nhau bằng dấu cách hoặc dấu tab.

Cách 2: Đưa ba số vào ba dòng khác nhau.

Cách 3: Hai số đầu cùng một dòng (cách nahu bởi dấu cách hoặ tab), số thứ ba trên dòng tiếp theo.

Cách 4: Số thứ nhất trên một dòng, hai số sau cùng một dòng tiếp theo (cách nhau bởi dấu cách hoặc tab), số thứ ba trên dòng tiếp theo. Khi vào sai sẽ báo lỗi và nhảy về chương trình chứa lời gọi nó.

Ví dụ minh họa sử dụng hàm printf, scanf nhập vào hai số a, b kiểu nguyên, tính toán và đưa kết quả biểu thức a^b, $\sqrt{a}$ lên màn hình.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
  int a, b;
  float kq;
  printf("\nNhap so thu nhat a= ");
```

```
  scanf("%d",&a);
  printf("\nNhap so thu hai b= ");
  scanf("%d",&b);
  kq=sqrt(a);
  printf("\nKet qua %d ^ %4d = %6.2f",a,b, pow(a,b));
  printf("\nKet qua can bac 2 cua %d = %4.2f",a, kq);
  getch();
}
```

3.4. Câu lệnh điều kiện

3.4.1. Lệnh if-else: Toán tử if cho phép lựa chọn chạy theo một trong hai nhánh tuỳ thuộc vào sự bằng không và khác không của biểu thức. Nó có hai cách viết sau :

if (biểu thức)
khối lệnh 1;
/* Dạng một */
	if (biểu thức)
	khối lệnh 1;
	else
	khối lệnh 2 ;
	/* Dạng hai */

Hoạt động của biểu thức dạng 1 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau lệnh if trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện ngay các lệnh tiếp sau lệnh if trong chương trình.

Hoạt động của biểu thức dạng 2 :

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau khối lệnh 2 trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện khối lệnh 2 sau đó thực hiện tiếp các lệnh tiếp sau khối lệnh 2 trong chương trình.

Ví dụ :

Chương trình nhập vào hai số a và b, tìm max của hai số rồi in kết quả lên màn hình. Chương trình có thể viết bằng cả hai cách trên như sau :

```
#include "stdio.h"
main()
        {
                float a,b,max;
                printf("\n Cho a=");
```

```
            scanf("%f",&a);
            printf("\n Cho b=");
            scanf("%f",&b);
            max=a;
            if (b>max) max=b;
            printf(" \n Max cua hai so a=%8.2f  va b=%8.2f la Max=%8.2f",a,b,max);
        }
#include "stdio.h"
main()
        {
            float a,b,max;
            printf("\n Cho a=");
            scanf("%f",&a);
            printf("\n Cho b=");
            scanf("%f",&b);
            if (a>b) max=a;
            else max=b;
            printf(" \n Max cua hai so a=%8.2f  va b=%8.2f la Max=%8.2f",a,b,max);
        }
```

Sự lồng nhau của các toán tử if :

C cho phép sử dụng các toán tử if lồng nhau có nghĩa là trong các khối lệnh (1 và 2) ở trên có thể chứa các toán tử if - else khác. Trong trường hợp này, nếu không sử dụng các dấu đóng mở ngoặc cho các khối thì sẽ có thể nhầm lẫn giữa các if-else.

Chú ý là máy sẽ gắn toán tử else với toán tử if không có else gần nhất. Chẳng hạn như đoạn chương trình ví dụ sau :

```
        if ( n>0 )        /* if thứ nhất*/
            if ( a>b )        /* if thứ hai*/
                z=a;
            else
                z=b;
```

thì else ở đây sẽ đi với if thứ hai.

Đoạn chương trình trên tương đương với :

```
        if ( n>0 )        /* if thứ nhất*/
             {
                if ( a>b )        /* if thứ hai*/
```

```
                    z=a;
            else
                    z=b;
        }
```

Trường hợp ta muốn else đi với if thứ nhất ta viết như sau :

```
    if ( n>0 )      /* if thứ nhất*/
        {
            if ( a>b )      /* if thứ hai*/
                z=a;
        }
    else
        z=b;
```

3.4.2. Lệnh else-if :

Khi muốn thực hiện một trong n quyết định ta có thể sử dụng cấu trúc sau :

```
if ( biểu thức 1 )
 khối lệnh 1;
else if ( biểu thức 2 )
 khối lệnh 2;
.....
else if ( biểu thức n-1 )
 khối lệnh n-1;
else
 khối lệnh n;
```

Trong cấu trúc này, máy sẽ đi kiểm tra từ biểu thức 1 trở đi đến khi gặp biểu thức nào có giá trị khác 0.

Nếu biểu thức thứ i (1,2, ...n-1) có giá trị khác 0, máy sẽ thực hiện khối lệnh i, rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Nếu trong cả n-1 biểu thức không có biểu thức nào khác 0, thì máy sẽ thực hiện khối lệnh n rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Ví dụ :

Chương trình giải phương trình bậc hai.

```
#include "stdio.h"
main()
 {
    float a,b,c,d,x1,x2;
```

```
printf("\n Nhap a, b, c:");
scanf("%f%f%f",&a&b&c);
d=b*b-4*a*c;
if (d<0.0)
        printf("\n Phuong trinh vo nghiem ");
else if (d= =0.0)
        printf("\n Phuong trinh co nghiem kep x1,2=%8.2f",-b/(2*a));
else
   {
        printf("\n Phuong trinh co hai nghiem ");
        printf("\n x1=%8.2f",(-b+sqrt(d))/(2*a));
        printf("\n x2=%8.2f",(-b-sqrt(d))/(2*a));
   }
```

3.5. Câu lệnh lựa chọn-lệnh switch

Là cấu trúc tạo nhiều nhánh đặc biệt. Nó căn cứ vào giá trị một biểu thức nguyên để để chọn một trong nhiều cách nhảy.

Cấu trúc tổng quát của nó là :

```
switch ( biểu thức nguyên )
  {
        case n1: khối lệnh 1
        case n2: khối lệnh 2
        .......
        case nk: khối lệnh k
        [ default: khối lệnh k+1          ]
}
```

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch.

default là một thành phần không bắt buộc phải có trong thân của switch.

Sự hoạt động của toán tử switch phụ thuộc vào giá trị của biểu thức viết trong dấu ngoặc ( ) như sau :

Khi giá trị của biểu thức này bằng ni, máy sẽ nhảy tới các câu lệnh có nhãn là case ni.

Khi giá trị biểu thức khác tất cả các ni thì cách làm việc của máy lại phụ thuộc vào sự có mặt hay không của lệnh default như sau :

Khi có default máy sẽ nhảy tới câu lệnh sau nhãn default.

Khi không có default máy sẽ nhảy ra khỏi cấu trúc switch.

Chú ý :

Máy sẽ nhảy ra khỏi toán tử switch khi nó gặp câu lệnh break hoặc dấu ngoặc nhọn đóng cuối cùng của thân switch. Ta cũng có thể dùng câu lệnh goto trong thân của toán tử switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch.

Khi toán tử switch nằm trong thân một hàm nào đó thì ta có thể sử dụng câu lệnh return trong thân của switch để ra khỏi hàm này (lệnh return sẽ đề cập sau).

Khi máy nhảy tới một câu lệnh nào đó thì sự hoạt động tiếp theo của nó sẽ phụ thuộc vào các câu lệnh đứng sau câu lệnh này. Như vậy nếu máy nhảy tới câu lệnh có nhãn case ni thì nó có thể thực hiện tất cả các câu lệnh sau đó cho tới khi nào gặp câu lệnh break, goto hoặc return. Nói cách khác, máy có thể đi từ nhóm lệnh thuộc case ni sang nhóm lệnh thuộc case thứ ni+1. Nếu mỗi nhóm lệnh được kết thúc bằng break thì toán tử switch sẽ thực hiện chỉ một trong các nhóm lệnh này.

Ví dụ: Lập chương trình phân loại học sinh theo điểm sử dụng cấu trúc switch :

```
#include "stdio.h"
main()
	{
		int diem;
		tt: printf("\nVao du lieu :");
		printf("\n Diem =");
		scanf("%d",&diem);
		switch (diem)
		{
			case 0:
			case 1:
			case 2:
			case 3:printf("Kem\n");break;
			case 4:printf("Yeu\n");break;
			case 5:
			case 6:printf("Trung binh\n");break;
			case 7:
			case 8:printf("Kha\n");break;
			case 9:
			case 10:printf("Gioi\n");break;
			default:printf(Vao sai\n);
```

```
        }
        printf("Tiep tuc 1, dung 0 :")
        scanf("%d",&diem);
        if (diem= =1) goto tt;
        getch();
        return;
}
```

3.6. Câu lệnh lặp for

Toán tử for dùng để xây dựng cấu trúc lặp có dạng sau :

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

Lệnh hoặc khối lệnh ;

Toán tử for gồm ba biểu thức và thân for. Thân for là một câu lệnh hoặc một khối lệnh viết sau từ khoá for. Bất kỳ biểu thức nào trong ba biểu thức trên có thể vắng mặt nhưng phải giữ dấu ;.

Thông thường biểu thức 1 là toán tử gán để tạo giá trị ban đầu cho biến điều khiển, biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục chu trình, biểu thức ba là một toán tử gán dùng để thay đổi giá trị biến điều khiển.

Hoạt động của toán tử for :

Toán tử for hoạt động theo các bước sau :

B1: Xác định biểu thức 1

B2: Xác định biểu thức 2

Tuỳ thuộc vào tính đúng sai của biểu thức 2 để máy lựa chọn một trong hai nhánh:

Nếu biểu thức hai có giá trị 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau thân for. Nếu biểu thức hai có giá trị khác 0 (đúng), máy sẽ thực hiện các câu lệnh trong thân for.

Tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình.

Chú ý : Nếu biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này việc ra khỏi chu trình for cần phải được thực hiện nhờ các lệnh break, goto hoặc return viết trong thân chu trình.

Trong dấu ngoặc tròn sau từ khoá for gồm ba biểu thức phân cách nhau bởi dấu ;. Trong mỗi biểu thức không những có thể viết một biểu thức mà có quyền viết một dãy biểu thức phân cách nhau bởi dấu phảy. Khi đó các biểu thức trong mỗi phần được xác định từ trái sang phải. Tính đúng sai của dãy biểu thức được tính là tính đúng sai của biểu thức cuối cùng trong dãy này.

Trong thân của for ta có thể dùng thêm các toán tử for khác, vì thế ta có thể xây dựng các toán tử for lồng nhau.

Khi gặp câu lệnh break trong thân for, máy ra sẽ ra khỏi toán tử for sâu nhất chứa câu lệnh này. Trong thân for cũng có thể sử dụng toán tử goto để nhảy đến một ví trí mong muốn bất kỳ.

3.7. Câu lệnh while

- Cú pháp : while (biểu thức 1) lệnh 1 ;
- Nguyên tắc thực hiện :
+b1. Tính giá trị của biểu thức 1.
+b2. Nếu giá trị của biểu thức 1 sai (= 0) thì chương trình ra khỏi vòng while
+b3. Nếu giá trị của biểu thức đúng thì thực hiện lệnh 1 và quay lại bước 1(b1).
- Chú ý : Biểu thức 1 có thể gồm nhiều biểu thức nhưng tính đúng sai phụ thuộc vào biểu thức cuối cùng.
Ví dụ : Nhập 1 dãy số nguyên từ bàn phím

```
#include < stdio.h >
#include < conio.h >
main ()
{
  int dayso [ 10 ] ; int i = 0 ;
 while ( i < 10)
     {
        printf ( "\n Số thu %d : ", i ); scanf ( " %d", & dayso [i]);
        i ++ ;
     }
```

3.8. Câu lệnh do… while

- Cú pháp :

```
do {
         lệnh 1 ;
}while ( biểu thức 1 ) ;
```

- Nguyên tắc thực hiện :
+b1. Máy thực hiện câu lệnh 1 ;
+b2. Sau đó tính giá trị của biểu thức 1, nếu giá trị của biểu thức 1 sai thì chương trình thoát ra khỏi vòng lặp. Nếu giá trị của biểu thức 1 đúng thì quay lại bước 1.
Chú ý : - while : Điều kiện được kiểm tra trước, nếu đúng mới thực hiện.
- do while : câu lệnh được thực hiện trước khi kiểm tra. Câu lệnh thực hiện bao giờ ít nhất là 1 lần. (do while ngược với
Repeat until của Pascal : lệnh Do while sai thì dừng, còn lệnh repeat until đúng thì dừng).
-Biểu thức 1 có thể gồm nhiều biểu thức, tuy nhiên tính đúng sai căn cứ theo biểu thức cuối cùng.
* Ví dụ : tính pi với sai số eps = 1E - 4 , pi = 4 - 4/3 + 4/5 - 4/7 + ...eps

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

```
float pi, dau, i , eps, saiso ;
i=1.0; dau = -1; saiso = 1e -4 ;
pi = 4.0;
printf ( "\n đang xử lý vui lòng đợi !");
do
 {
   eps = 4.0 / ( 2.0 * i + 1.0 );
   pi + = dau * eps ; dau = dau * - 1.0 ; i + = 1.0;
 }
  while ( eps > saiso );
     printf ("\n số pi là : " % f ", pi ) ;
     getch ();
}
```

3.9. Câu lệnh break

- Cú pháp : Dùng để thoát khỏi vòng lặp. Khi gặp câu lệnh này trong vòng lặp, máy ra khỏi và chỉ đến câu lệnh sau các lệnh trên. Nếu nhiều vòng lặp ----> break sẽ thoát ra khỏi vòng lặp gần nhất.

3.10. Lệnh continue

- Cú pháp continue; : khi gặp lệnh này trong các vòng lặp, máy sẽ bỏ qua phần còn lại trong vòng lặp và tiếp tục thực hiện vòng lặp tiếp theo.
- Đối với lệnh For máy sẽ tính lại biểu thức 3 (bt3) và quay lại bước 2.
- Đối với lệnh while, do while máy sẽ tính lại giá trị của biểu thức 1 và quay lại bước 1.

* Ví dụ : Nhập 1 chuỗi ký tự kể cả ký tự trống và bỏ qua các ký tự không hợp lệ và kết thúc khi ấn ESC hoặc số ký tự vượt quá kích thước mảng.

```
char xau [MAXL], kytu ;
int i = 0 ;
while (1) /* luôn luôn đúng vòng lặp vĩnh cửu */
  {
   kytu = getch ( ) ;
   if ( kytu = = 27 ) break ;
    if ( i >= MAXL ) break ;
    if ( kytu > 122 || kytu < 65 ) continue ;
       Xau [ i ++] = kytu ;
  }
xau [ i ] = ' \ 0 ' ;
```

3.11. Toán tử goto và nhãn (label)

- Ví dụ : tiep tuc : st = a[i]; => tiep tuc là nhãn của lệnh st = a [i];
- Lệnh goto nhãn => nhảy đến câu lệnh đứng sau nhãn.
- CHÚ Ý : PHẠM VI NHÃN TRONG CÙNG 1 HÀM.

Bài tập:

1. Viết chương trình giải pt bậc nhất ax+b=0
2. Viết chương trình giải bất pt bậc nhất ax+b<0
3. Viết chương trình giải pt bậc hai $ax^2+bx+c=0$

4. Viết chương trình tìm ước số chung lớn nhất của 2 số
5. Viết chương trình tìm số lớn nhất trong dãy các số nhập vào

CHƯƠNG 4. HÀM CHƯƠNG TRÌNH VÀ CẤU TRÚC CHƯƠNG TRÌNH.

4.1. Khái niệm về chương trình con

Một chương trình viết trong ngôn ngữ C là một dãy các hàm, trong đó có một hàm chính (hàm main()). Hàm chia các bài toán lớn thành các công việc nhỏ hơn, giúp thực hiện những công việc lặp lại nào đó một cách nhanh chóng mà không phải viết lại đoạn chương trình. Thứ tự các hàm trong chương trình là bất kỳ, song chương trình bao giờ cũng đi thực hiện từ hàm main().

4.2. Hàm trong C

Hàm có thể xem là một đơn vị độc lập của chương trình. Các hàm có vai trò ngang nhau, vì vậy không có phép xây dựng một hàm bên trong các hàm khác.

Xây dựng một hàm bao gồm: khai báo kiểu hàm, đặt tên hàm, khai báo các đối và đưa ra câu lệnh cần thiết để thực hiện yêu cầu đề ra cho hàm. Một hàm được viết theo mẫu sau :

```
type tên hàm ( khai báo các đối )
{
        Khai báo các biến cục bộ
        Các câu lệnh
        [return[biểu thức];]
}
```

Dòng tiêu đề :

Trong dòng đầu tiên của hàm chứa các thông tin về : kiểu hàm, tên hàm, kiểu và tên mỗi đối.

Ví dụ :

```
        float max3s(float a, float b, float c)
```

khai báo các đối có dạng :

Kiểu đối 1 tên đối 1, kiểu đối 2 tên đối 2,..., kiểu đối n tên đối n

Thân hàm :

Sau dòng tiêu đề là thân hàm. Thân hàm là nội dung chính của hàm bắt đầu và kết thúc bằng các dấu { }. Trong thân hàm chứa các câu lệnh cần thiết để thực hiện một yêu cầu nào đó đã đề ra cho hàm.

Thân hàm có thể sử dụng một câu lệnh return, có thể dùng nhiều câu lệnh return ở các chỗ khác nhau, và cũng có thể không sử dụng câu lệnh này.

Dạng tổng quát của nó là :

return [biểu thức];

Giá trị của biểu thức trong câu lệnh return sẽ được gán cho hàm.

Ví dụ :

Xét bài toán: Tìm giá trị lớn nhất của ba số mà giá trị mà giá trị của chúng được đưa vào bàn phím. Xây dựng chương trình và tổ chức thành hai hàm : Hàm main() và hàm max3s. Nhiệm vụ của hàm max3s là tính giá trị lớn nhất của ba số đọc vào, giả sử là a,b,c. Nhiệm vụ của hàm main() là đọc ba giá trị vào từ bàn phím, rồi dùng hàm max3s để tính như trên, rồi đưa kết quả ra màn hình.

Chương trình được viết như sau :

```
#include "stdio.h"
float max3s(float a,float b,float c ); /* Nguyên mẫu hàm*/
main()
        {
                float x,y,z;
                printf("\n Vao ba so x,y,z:");
                scanf("%f%f%f",&x&y&z);
                printf("\n Max cua ba so x=%8.2f y=%8.2f z=%8.2f la : %8.2f",
                x,y,z,max3s(x,y,z));
        }       /* Kết thúc hàm main*/

float max3s(float a,float b,float c)
        {
                float max;
                max=a;
                if (max<b) max=b;
                if (max<c) max=c;
                return(max);
        } /* Kết thúc hàm max3s*/
```

Quy tắc hoạt động của hàm:

Một cách tổng quát lời gọi hàm có dạng sau :

tên hàm ([Danh sách các tham số thực])

Số các tham số thực tế thay vào trong danh sách các đối phải bằng số tham số hình thức và lần lượt chúng có kiểu tương ứng với nhau.

Khi gặp một lời gọi hàm thì nó sẽ bắt đầu được thực hiện. Nói cách khác, khi máy gặp lời gọi hàm ở một vị trí nào đó trong chương trình, máy sẽ tạm dời chỗ đó và chuyển đến hàm tương ứng. Quá trình đó diễn ra theo trình tự sau :

Cấp phát bộ nhớ cho các biến cục bộ.

Gán giá trị của các tham số thực cho các đối tương ứng.

Thực hiện các câu lệnh trong thân hàm.

Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các đối, biến cục bộ và ra khỏi hàm.

Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

4.3. Chuyển tham số cho hàm

Các tham số hình thức: là các tham số được khai báo ở phần khai báo (định nghĩa hàm). Các tham số này không có giá trị thực sự, chúng chỉ có ý nghĩa về mặt hình thức.

Các tham số thực: là các tham số được truyền vào khi hàm được gọi thực hiện. Chúng thay thế các tham số hình thức.

4.4. Biến toàn cục và biến địa phương

Biến toàn cục là biến nằm ngoài tất cả các hàm. Biến cục bộ là biến được khai báo bên trong một hàm nào đó. Tham số hình thức có thể coi là biến cục bộ.

Do đối và biến cục bộ đều có phạm vi hoạt động trong cùng một hàm nên đối và biến cục bộ cần có tên khác nhau.

Đối và biến cục bộ đều là các biến tự động. Chúng được cấp phát bộ nhớ khi hàm được xét đến và bị xoá khi ra khỏi hàm nên ta không thể mang giá trị của đối ra khỏi hàm.

Đối và biến cục bộ có thể trùng tên với các đại lượng ngoài hàm mà không gây ra nhầm lẫn nào.

Khi một hàm được gọi tới, việc đầu tiên là giá trị của các tham số thực được gán cho các đối (trong ví dụ trên hàm max3s, các tham số thực là x,y,z, các đối tương ứng là a,b,c). Như vậy các đối chính là các bản sao của các tham số thực. Hàm chỉ làm việc trên các đối.

Các đối có thể bị biến đổi trong thân hàm, còn các tham số thực thì không bị thay đổi.

Chú ý :

Khi hàm khai báo không có kiểu ở trước nó thì nó được mặc định là kiểu int.

Không nhất thiết phải khai báo nguyên mẫu hàm. Nhưng nói chung nên có vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi hàm hay tự động việc chuyển dạng.

Nguyên mẫu của hàm thực chất là dòng đầu tiên của hàm thêm vào dấu ;. Tuy nhiên trong nguyên mẫu có thể bỏ tên các đối.

Hàm thường có một vài đối. Ví dụ như hàm max3s có ba đối là a,b,c. cả ba đối này đều có giá trị float. Tuy nhiên, cũng có hàm không đối như hàm main.

Hàm thường cho ta một giá trị nào đó. Lẽ dĩ nhiên giá trị của hàm phụ thuộc vào giá trị các đối.

Hàm không cho các giá trị

Các hàm không cho giá trị giống như thủ tục (procedure) trong ngôn ngữ lập trình PASCAL. Trong trường hợp này, kiểu của nó là void.

Ví dụ hàm tìm giá trị max trong ba số là max3s ở trên có thể được viết thành thủ tục hiển thị số cực đại trong ba số như sau :

```
void htmax3s(float a, float b, float c)
        {
                float max;
                max=a;
                if (max<b) max=b;
                if (max<c) max=c;
        }
```

Lúc này, trong hàm main ta gọi hàm htmax3s bằng câu lệnh :

```
htmax3s(x,y,z);
```

4.5.Tính đệ quy của hàm

4.5.1. Mở đầu: C không những cho phép từ hàm này gọi tới hàm khác, mà nó còn cho phép từ một điểm trong thân của một hàm gọi tới chính hàm đó. Hàm như vậy gọi là hàm đệ qui.

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 hoạ 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 gt(int n) /* Tính n! với n>=0*/
 {
        long int gtphu=1;
        int i;
        for (i=1;i<=n;++i)
        gtphu*=i;
        return s;
 }
```

Ta nhận thấy rằng n! có thể tính theo công thức truy hồi sau :

n!=1 nếu n=0

n!=n*(n-1)! nếu n>0

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

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

Ta đi giải thích hoạt động của hàm đệ qui khi sử dụng trong hàm main dưới đây :

```
#include "stdio.h"
main()
    {
      printf("\n 3!=%d",gtdq(3));
    }
```

Lần gọi đầu tiên tới hàm gtdq được thực hiện từ hàm main(). Máy sẽ tạo ra một tập các biến tự động của hàm gtdq. 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*gtdq(n-1) (*)

Để tính biểu thức trên, máy cần gọi chính hàm gtdq 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*gtdq(n-1) (**)

Biểu thức trên lại gọi hàm gtdq 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=gtdq(1) (***)

Bắt đầu từ đây, máy sẽ thực hiện ba lần ra khỏi hàm gtdq. Lần ra khỏi hàm thứ nhất ứng với lần vào thứ ba. Kết quả là đối n thứ ba được giải phóng, hàm gtdq(1) cho giá trị là 1 và máy trở về xét giá trị biểu thức

n*gtdq(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*gtdq(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

Chú ý :

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.

4.5.2. Các bài toán có thể dùng đệ qui

Phương pháp đệ qui thường áp dụng cho các bài toán phụ thuộc tham số có hai đặc điểm sau :

Bài toán dễ dàng giải quyết trong một số trường hợp riêng ứng với các giá trị đặc biệt của tham số. Người ta thường gọi là trường hợp suy biến.

Trong trường hợp tổng quát, bài toán có thể qui về một bài toán cùng dạng nhưng giá trị tham số thì bị thay đổi. Sau một số hữu hạn bước biến đổi dệ qui nó sẽ dẫn tới trường hợp suy biến.

Bài toán tính n giai thừa nêu trên thể hiện rõ nét đặc điểu này.

4.5.3. Cách xây dựng hàm đệ qui :

Hàm đệ qui thường được xây dựng theo thuật toán sau :

```
if ( trường hợp suy biến)
        {
                Trình bày cách giải bài toán khi suy biến
        }
else /* Trường hợp tổng quát */
        {
                Gọi đệ qui tới hàm ( đang viết ) với các giá
                trị khác của tham số
        }
```

4.5.4. Các ví dụ về dùng hàm đệ qui :

Ví dụ 1 :

Bài toán dùng đệ qui tìm USCLN của hai số nguyên dương a và b.

Trong trường hợp suy biến, khi a=b thì USCLN của a và b chính là giá trị của chúng.

Trong trường hợp chung :

uscln(a,b)=uscln(a-b,b) nếu a>b

uscln(a,b)=uscln(a,b-a) nếu a<b

Ta có thể viết chương trình như sau :

```
#include "stdio.h"
int uscln(int a,int b ); /* Nguyên mẫu hàm*/
main()
        {       int m,n;
                printf("\n Nhap cac gia tri cua a va b :");
                scanf("%d%d",&m,&n);
                printf("\n USCLN cua a=%d va b=%d la :%d",m,m,uscln(m,n))
        }
int uscln(int a,int b)
        {
                if (a==b)
                        return a;
                else
                  if (a>b)
                        return uscln(a-b,b);

                  else
                        return uscln(a,b-a);
        }
```

Ví dụ 2 :

Chương trình đọc vào một số rồi in nó ra dưới dạng các ký tự liên tiếp.

```
# include "stdio.h"
# include "conio.h"
void prind(int n);
main()
```

```
{
  int a;
  clrscr();
  printf("n=");
  scanf("%d",&a);
  prind(a);
  getch();
}
void prind(int n)
 {
  int i;
  if (n<0)
  { putchar('-');
    n=-n;
  }
  if ((i=n/10)!=0)
  prind(i);
  putchar(n%10+'0');
 }
```

4.6. Bộ tiền xử lý C

C đưa ra một số cách mở rộng ngôn ngữ bằng các bộ tiền sử lý macro đơn giản. Có hai cách mở rộng chính là #define mà ta đã học và khả năng bao hàm nội dung của các file khác vào file đang được dịch.

Bao hàm file :

Để dễ dàng xử lý một tập các #define và khai báo (trong các đối tượng khác), C đưa ra cách bao hàm các file khác vào file đang dịch có dạng :

#include "tên file"

Dòng khai báo trên sẽ được thay thế bởi nội dung của file có tên là tên file. Thông thường có vài dòng như vậy xuất hiện tại đầu mỗi file gốc để gọi vào các câu lệnh #define chung và các khai báo cho các biến ngoài. Các #include được phép lồng nhau. Thường thì các #include được dùng nhiều trong các chương trình lớn, nó đảm bảo rằng mọi file gốc đều được cung cấp cùng các định nghĩa và khai báo biến, do vậy tránh được các lỗi khó chịu do việc thiếu các khai báo định nghĩa. Tất nhiên khi thay đổi file được bao hàm vào thì mọi file phụ thuộc vào nó đều phải dịch lại.

Phép thế MACRO :

Định nghĩa có dạng :

#define biểu thức 1 [biểu thức 2]

sẽ gọi tới một macro để thay thế biểu thức 2 (nếu có) cho biểu thức 1.

Ví dụ :

#define YES 1

Macro thay biến YES bởi giá trị 1 có nghĩa là hễ có chỗ nào trong chương trình có xuất hiện biến YES thì nó sẽ được thay bởi giá trị 1.

Phạm vi cho tên được định nghĩa bởi #define là từ điểm định nghĩa đến cuối file gốc. Có thể định nghĩa lại tên và một định nghĩa có thể sử dụng các định nghĩa khác trước đó. Phép thế không thực hiện cho các xâu dấu nháy, ví dụ như YES là tên được định nghĩa thì không có việc thay thế nào được thực hiện trong đoạn lệnh có "YES".

Vì việc thiết lập #define là một bước chuẩn bị chứ không phải là một phần của chương trình biên dịch nên có rất ít hạn chế về văn phạm về việc phải định nghĩa cái gì. Chẳng hạn như những người lập trình ưa thích PASCAL có thể định nghĩa :

```
#define then
#define begin {
#define end; }
```

sau đó viết đoạn chương trình :

```
if (i>0) then
        begin
                a=i;
                .....
        end;
```

Ta cũng có thể định nghĩa các macro có đối, do vậy văn bản thay thế sẽ phụ thuộc vào cách gọi tới macro.

Ví dụ :

Định nghĩa macro gọi max như sau :

```
#define max(a,b) ((a)>(b) ?(a):(b))
```

Việc sử dụng :

```
x=max(p+q,r+s);
```

tương đương với :

```
x=((p+q)>(r+s) ? (p+q):(r+s));
```

Như vậy ta có thể có hàm tính cực đại viết trên một dòng. Chừng nào các đối còn giữ được tính nhất quán thì macro này vẫn có giá trị với mọi kiểu dữ liệu, không cần phải có các loại hàm max khác cho các kiểu dữ liệu khác nhưng vẫn phải có đối cho các hàm.

Tất nhiên nếu ta kiểm tra lại việc mở rộng của hàm max trên, ta sẽ thấy rằng nó có thể gây ra số bẫy. Biểu thức đã được tính lại hai lần và điều này là không tốt nếu nó gây ra hiệu quả phụ kiểu như các lời gọi hàm và toán tử tăng. Cần phải thận trọng dùng thêm dấu ngoặc để đảm bảo trật tự tính toán. Tuy vậy, macro vẫn rất có giá trị.

Chú ý :

Không được viết dấu cách giữa tên macro với dấu mở ngoặc bao quanh danh sách đối.

Ví dụ :

Xét chương trình sau :

```
main()
        {
                int x,y,z;
                x=5;
                y=10*5;
                z=x+y;
                z=x+y+6;
                z=5*x+y;
                z=5*(x+y);
                z=5*((x)+(y));
                printf("Z=%d",z);
                getch();
                return;
        }
```

Chương trình sử dụng MACRO sẽ như sau :

```
#define BEGIN {
#define END }
#define INTEGER int
#define NB 10
#define LIMIT NB*5
#define SUMXY x+y
#define SUM1 (x+y)
#define SUM2 ((x)+(y))
main()
```

```
BEGIN
        INTEGER x,y,z;
        x=5;
        y=LIMIT;
        z=SUMXY;
        z=5*SUMXY;
        z=5*SUM1;
        z=5*SUM2;
        printf("\n Z=%d",z);
        getch();
        return;
END
```

Bài tập

1. Viết hàm tìm ước số chung lớn nhất của 2 số
2. Viết hàm tìm bội số chung nhỏ nhất của 2 số
3. Viết hàm tìm số lớn nhất của 3 số
4. Viết hàm in các số chẵn trong một dãy số nhập từ bàn phím
5. Viết hàm tính tổng các số từ 1 - n

CHƯƠNG 5. MẢNG VÀ CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC

5.1.Dữ liệu kiểu mảng/con trỏ

Mảng là tập hợp của các biến cùng kiểu được xếp liên tiếp nhau trong bộ nhớ trong.

5.1.1. Mảng 1 chiều và nhiều chiều

a/ Khai báo mảng 1 chiều : < kiểu phần tử > < tên mảng> [< chỉ số >]
Ví dụ : int a [5] ; => a [0] a[1] a[2] a [3] a [4] (chỉ số chạy từ 0 đến n - 1).
char S [20] ; => 'A' 'B' ..... 'X '
S[0]S[1] S[19]
b/ Cách nhập số liệu cho mảng từ bàn phím (có thể dùng hàm Random C).
+ Mảng số nguyên :
Ví dụ : Nhập vào mảng số nguyên 5 phần tử

```
#include < stdio.h>
#include < conio.h>
#define n 5
main ()
{
  int a [ n ] ; int i ;
  for ( i = 0 ; i < n ; i ++ )
    {
     printf ( " a [ %d ] = " , i ); scanf ( " % d" , & a [ i ]);
    }
/* Xuất số liệu mảng ra màn hình */
    for ( i = 0 ; i < n ; ++ i)
    printf ( " \ n a [ % d ] = % d ", i , a [ i ]);
    getch ();
}
```

+ Mảng số thực float :

```
#include <stdio.h>
#include < conio.h>
#define n 5 ;
main ()
{
   float a [ n ] , tam ;
   .....scanf ( " % f " , &tam) ; /*nhập qua biến trung gian tạm */
   a [ i ] = tam ;
```

c/Khởi tạo mảng :
a [5] = { 1,2,3,5,4 }a[0]=1 a[2]=2 .. a[4]=4
d/ Mảng ký tự :
- là chuỗi ký tự kết thúc bằng ký tự NULL có mã ASCII là 0 .
- Ví dụ : char S [3] = { 'L', '0', 'P'] : chuỗi này không đúng do thiếu chỗ cho ký tự kết thúc là NULL.
- Ta có thể gán :
char S [4] = " Lop "; Ngôn ngữ C sẽ tự động ghi ký tự kết thúc là NULL, tức là ' \0 '.
char S[ ] = " Lop " ; Không cần khai báo số phần tử mảng.
* Ví dụ 1 : Nhập vào một mảng số nguyên sau đó sắp xếp theo thứ tự tăng dần :

```
#include < stdio.h>
#define n 5
main ( )
  {
```

```
    int a [ n ] ; int i , j, t ;
    for ( i = 0 ; i > n ; i ++ );
      {
        printf ( " nhập a [ % d] = " , i ); scanf ( " %d", & a [i ]);
      }
    /* Sắp xếp tăng dần */
      for ( i = 0 ; i < n - 1 ; i ++)
      for ( j = i + 1 ; j < n ; j ++ )
        if ( a [ i ] < a [j ] )
          {
            t = a [ i ] ; a [ i ] = a [ j ]; a [j ] = t ;
          }
/* in kết quả */
      for ( i = 0 ; i < n ; i ++ )
      printf ( " % 5d " , a [ i ] );
      getch ( );
   }
```

Ví dụ 2 : Làm lại ví dụ 1 nhưng viết riêng hàm sắp xếp và truyền tham số cho mảng 1 chiều

```
#include <stdio.h>
#include <conio.h>
#define N 5
void sapxep ( int a [ ] , int n );
void main ( )
  {
    int a [ N ] ; int i ;
   /* nhập 1 số liệu cho mảng */
    for ( i = 0 ; i < N , i ++ )
    {
       printf ( " A [ %d ] = ", i ); scanf ( " %d ", & a [ i ] ); }
  /* gọi hàm sắp xếp để sắp tăng dần */
         sapxep ( a, N );
/* in kết quả */
        for ( i = 0 ; i < N ; i ++ )
           printf ( " %5d ", a [ i ] );
           getch ( );
   }
/* hàm sắp xếp tăng dần */
void sapxep ( int a [ ], int n )
  {
     int i, j, t ;
      for ( i = 0 ; i > n - 1 ; i ++)
      for ( j = i + 1 ; j < n ; j ++ )
        if ( a [ i ] > a [ j ]
           {
              t = a [ i ] ; a [ i ] = a [ j ] ; a [j ] = t ;
           }
```

* Ví dụ 3 : chuyển đổi 1 chuỗi ký tự thường thành Hoa.

Chú ý : + Hàm tolower (ch) : đổi 1 ký tự ch thành thường.

+ Hàm toupper (ch) : đổi ký tự ch thành Hoa.

+ Cả 2 hàm trên đều năm trong thư viện : < ctyte.h>

Giải : #include < stdio.h>

include < ctyte.h>

```
#define n 20
main ( )
   {
      char s [ n ] ; int i ;
      for ( i = 0 ; i < n ; i ++ )
      s[ i ] = toupper ( getchar ( ) ) ; /* nhập ký tự và đổi thành hoa lưu vào mảng */
/* kết xuất chuỗi s */
     for ( i = 0 ; i < n ; i ++ )
     putchar ( s [ i ] ) ; /* putchar ( ch ) : in ký tự ch ra màn hình */
     getch ( )
  }
```

Bài tập : 1/ viết chương trình nhập số liệu cho mảng A gồm N phần tử và mảng B gồm n phần tử , sau đó ghép 2 mảng A và B thành mảng C gồm m + n phần tử và sắp xếp tăng dần (Bài này phải dùng hàm nhập số liệu cho mảng và hàm sắp xếp).

- Tính tổng các phần tử âm, dương, số chẵn, số lẽ và tổng tất cả các phần tử của mảng C [m + n].In các số lẻ trên 1 hàng và các số chẵn trên 1 hàng.
- Nhập vào một giá trị và tìm xem giá trị đó có thuộc vào mảng C không. Nếu có in ra tất cả các phần tử tìm được.

d) Mảng nhiều chiều

Khai báo mảng hai chiều : < kiểu phần tử > < tên mảng > [< chỉ số hàng >] [< chỉ số cột >]

*Ví dụ 1 : int a [3] [2] ; float b [3] [4] ; char c [5] [6] ;

=> a [0] [0] a [0] [1]

a [1] [0] a [1] [1]

a [2] [0] a [2] [1]

Ví dụ 2 : #define Hang 5

define Cot 6

int a [Hang] [Cot] ;

=> ta có các biến chạy i (chỉ số chạy từ 0 đến (Dong - 1)).

ta có các biến chạy j (chỉ số chạy từ 0 đến (Cot - 1)) .

a [0] [0] a [0][1] ..... a [0][Cot - 1]

a [1] [0] a [1][1] ..... a [a][Cot - 1]

. . . . . .

a[Dong-1][0]..... . . . . . a[Dong-1][Cot-1]

*Ví dụ : Viết chương trình tính tổng, tích các số trong mảng số thực a[3][2] ;

```
#include < stdio.h>
#define N 3
#define N 2
main ( )
{
   int i , j ; float a [M][N] ; float tong, tich, tam ;
/* nhập số liệu */
   for ( i = 0 ; i < M ; i ++ )
    for ( j = 0 ; j < N ; j ++ )
       { printf ( " nhập a [ %d][%d] = " , i , j );
         scanf ( " %f " , & tam ) ; a [i][j] = tam ;}
/* tính tổng */
      Tong = 0 ; Tich = 1;
      for ( i = 0 ; i < M ; i ++ )
      for ( j = 0 ); j < N ; j ++ )
```

```
      {
          Tong = Tong + a [ i ][j] ; Tich = Tich * a [i][j] ; }
/* in kết quả */
printf ( " Tổng là tổng = %f, TONG );
printf ( " tích là TICH = %F, TICH );
getch ( ) ;
}
```
Truyền tham số mảng nhiều chiều cho hàm (tham số thực là tên mảng nhiều chiều)
- giả sử a là mảng 2 chiều : float a[M][N]
+ Chương trình gọi :
```
{

float a [M][N]
Tong ( a ) ;// ( truyền địa chỉ của mảng cho hàm )
}
```
+ Chương trình bị gọi (chương trình con) :
```
float tong ( float a[ ][N] ) /* khai báo đối để nhận địa chỉ của mảng */
{
}
```
Note : hàm tong chỉ dùng được đối với các mảng hai chiều có N cột và số hàng không quan trọng, không khai báo) :
* Ví dụ : Viết chương trình tính tổng của 2 ma trận cấp m x n theo công thức :
C[i][j] = a[i][j] + b [i][j]
```
#include <stdio.h>
#define m 3
#define n 4
/* các prototype ( khai báo hàm )*/
void nhap ( int a[ ][N] , int M, int N );
void TongMT ( int a[ ][N], int b[ ][N] , int c [ ][N], int M , int N );
void InMT ( int c [ ][N], int M, int N );
/* chương trình chính */
{ int a [M][N], b[M][N], c[M][N] ;
/* gọi các hàm */
Nhap ( a, M ,N ) ; nhap ( b, M,N);
TONGMT ( a, b, c , M, N );
InMT ( c, M, N );
Getch ( ) ;
}
/* Hàm nhập số liệu cho mảng 2 chiều m x n phần tử */
void Nhap ( int a [ ][N] , int M , int N )
{
  int i , j ;
  for ( i= 0 ; i < M ; i ++ )
  for ( j = 0 ; j < N ; j++ )
    {
        printf ( " a[%d][5d] = " , i , j ) ; scanf ( " %d " , &a [i][j]) ; }
    return ;
}
void TongM ( int a [ ][N], int b [ ][N], int c [ ][N], int M , int N )
{
  int i, j ;
  for ( i = 0 ; i < M ; i ++ )
```

```
  for ( j = 0 ; j < N ; j ++ )
      c [i][j] = a [i][j] + b [i][j] ;
   return ;
}
/* in kết quả */
void inMT ( int c[ ][N], int M, int N )
{
  int i, j ;
  for ( i = o ; i < M ; i ++ )
    { for ( j = 0 ; j < N ; j ++ )
       printf ( " % 3d", a[i][j] );
       printf ( " \n " ) ; /* xuống dòng */
    }
   return ;
}
```

Bài tập mảng :

1/ cho mảng 2 chiều A, là ma trận vuông cấp n x n , lập chương trình :

a/ tính tổng tất cả các phần tử dương của mảng.

b/ tính tổng các phần tử A[i][j] mà i + j chia hết cho 5 .

c/ In ra các số nguyên tố theo từng hàng.

d/ Sắp xếp theo hàng.

e/ Sắp xếp theo cột .

f/ Tính tổng các phần tử trên đường chéo (i = j) , đường biên.

g/ Tìm max ; min theo từng hàng, cột và toàn bộ ma trận.

2/ Một chuỗi gọi là palindrone nếu nó không thay đổi khi ta đảo ngược thứ tự của các ký tự trong nó (ví dụ " 12321 ")

. Lập chương trình đọc một chuỗi (xâu) ký tự và xác định xem có tính palondrone không.

5.1.2. Biến con trỏ

a) Khái niệm con trỏ (pointer) và địa chỉ :

- Mỗi biến trong ngôn ngữ C đều có 1 tên và tương ứng với nó là một vùng nhớ dùng để chứa giá trị của nó. Tuỳ theo biến mà vùng nhớ dành cho biến có độ dài khác nhau. Địa chỉ của biến là sô thứ tự của byte đầu tiên tương ứng với biến đó. Địa chỉ của biến có kiểu khác nhau là khác nhau. Địa chỉ và biển kiểu int liên tiếp cách nhau 2 byte , biến kiểu float là 4 byte.
- Con trỏ là biến dùng để chứa địa chỉ của biến khác hoặc có thể là một hàm. Do có nhiều loại địa chỉ nên cũng có nhiều loại biến con trỏ. Con trỏ kiểu int dùng để chứa địa chỉ của kiểu int. Con trỏ kiểu float dùng để chứa địa chỉ kiểu float.
- Muốn sử dụng được pointer, trước tiên phải có được địa chỉ của biến mà ta cần quan tâm bằng phép toán lấy địa chỉ & . Kết quả của phép lấy địa chỉ & sẽ là 1 phần tử hằng.

* Ví dụ : int num ; => &num là địa chỉ của num.

```
int pnum ; /* pnum là 1 pointer chỉ đến một int */
pnum = & num ; /* pnum chứa địa chỉ biến int num*/
giả sử : num = 5 ; => * pnum = 5 /* do * là toán tử nội dung */
Hai câu lệnh sau đây là tương đương
Num = 100 ;
( * pnum ) = 100 ;
```

- Quy tắc khai báo biến con trỏ : < kiểu dữ liệu> * < tên biến con trỏ >

*Ví dụ 2 : int a, *p ;

```
a = 5 ; /* giả sử địa chỉ của a là < 106 > */
p = & a ; /* p = <106> */
p = a ; /* phép gán sai */
```

* p = a ; /* phép gán đúng */
scanf (" %d " , &a) ; tương đương scanf (" %d , p) ;
b) Tính toán trên biến con trỏ (pointer)
+ Hai biến con trỏ cùng kiểu có thể gán cho nhau :
Ví dụ 1 : int a, * p, *a ; float * f;
a = 5 ; p = &a ; q = p ; /* đúng */
f = p ; /* sai do khác kiểu */
f = (float *)p ; /* đúng nhờ ép kiểu con trỏ nguyên về kiểu float */
Ví dụ 2 : int a ;
char *c ;
c = &a ; /* sai vì khác kiểu */
c = (char*) /* đúng */
+ Một biến pointer có thể được cộng, trừ với một số nguyên (int , long) để cho kết quả là một pointer.
* Ví dụ : int a , *p , * p10 ;
a = 5 ;
p = &a ;
p10 = p + 10 ;
Ví dụ : int V[10] ;/* mảng 10 phần tử */
int *p ;
p = & V[0];
for (i = 0 ; i < 10 ; i ++)
{ *p = i ; /* gán giá trị i cho phần tử mà p đang trỏ đến */
p ++ /* p được tăng lên 1 để chỉ đến phần tử kế tiếp */
}
/* kết quả V[0] = 0 , V [1] = 1 ... V[9] = 9 * /
+ Phép trừ 2 pointer cho kết quả là một số int biểu thị khoảng cách (số phần tử) giữa 2 pointer đó.
+ Phép cộng 2 pointer là không hợp lệ, pointer không được nhân chia với 1 số nguyên hoặc nhân chia vơi nhau.
+ p = NULL : là con trỏ p không trỏ đến đâu cả.
Chú ý : không được sử dụng biến con trỏ khi chưa được khởi gán .
Ví dụ : int a , *p ;
scanf ("%d", p) (sai)
=> thay bằng các lệnh : p = &a và scanf ("%d" p) (đúng)

c) Con trỏ mảng

+ Mảng 1 chiều và con trỏ :
- Trong ngôn ngữ C : giữa mảng và con trỏ có mối quan hệ chặt chẽ. Các phần tử của mảng có thể xác định nhờ chỉ số hoặc thông qua con trỏ.
- Ví dụ : int A[5] ; * p ;
P = A ;
+ mảng bố trí 5 ô nhớ liên tiếp (mỗi ô chiếm 2 byte).
+ Tên mảng là 1 hằng địa chỉ (không thay đổi được), chính là địa chỉ của phần tử đầu tiên.
=> A tương đương với &A[0]
(A + i) tương đương với &A[i]
*(A + i) tương đương với A[i]
p = A => p = &A[0] (p trỏ tới phần tử A[0])
*(p + i) tương đương với A[i].
=>bốn cách viết như sau là tương đương : A[i], * (a + i), * (p + i), p[i].

Ví dụ 2 : int a [5] ; *p ;
p = a ;
for (i = 0; i < 5 ; ++ i)
scanf (" %d ", &a[i]); (1)
scanf (" %d ",a + i); (2)
scanf (" %d", p + i); (3)
scanf (" % d", p ++); (4)
scanf (" %d ", a ++); sai vì địa chỉ của a là hằng.
- Các lệnh (1), (2), (3), (4) tương đương nhau.
Ví dụ 3 : Nhập 5 số nguyên vào 1 mảng gồm 5 phần tử (a[5]) sau đó sắp xếp tăng dần, in ra số lớn nhất vf nhỏ nhất và tính tổng của 5 số đó.

```
#include <stdio.h>
#define n 5
main ( )
  { int a [n], t , *p, i , j, ; int s ;
     p = a ;
     for ( i = 0; i < n ; i ++ )
      { printf ( " a[%d] = " , i ) ; scanf ( " %d ", p + i ) }
/* Sắp xếp tăng dần */
     for ( i = 0 ; i < n-1 ; i ++ )
     for ( j = i + 1 ; j<n ; j++)
        if ( *(a + i ) > * ( a + j )
           { t = * ( a + i ) ; *(a + i ) = * ( a + j) ; *(a + j ) = t ; }
      s= 0 ;
     for ( j=0 ; i < n , ++i )
       s + = a[ i];
        printf ("\n Tong = %5d ", s );
        printf ( "\n số lớn nhất là %d ", a [4] );
        printf ( " số nhỏ nhất là %d \n ", a [d] );
       getch ( );
      }
```

+ Con trỏ và mảng nhiều chiều :
- Phép toán lấy địa chỉ & chỉ áp dụng được với mảng 2 chiều kiểu nguyên. Các kiểu khác không được.
* Ví dụ 1 : int a[2][3]
{ scanf ("%d", & a[1][1]) } (đúng)
* Ví dụ 2 : float a[2][3]
Scanf (" %f", &a[1][1]); (sai).
- Mảng 2 chiều a[2][3] => gồm 2 x 3 = 6 phần tử có 6 địa chỉ liên tiếp theo thứ tự sau :
Phần tử : a[0][0] a[0][1] a[0][2] a[1][0] a[1][1] a[1][2] (*)
Địa chỉ : 0 1 2 3 4 5
- Ngôn ngữ C quan niệm mảng 2 chiều là mảng một chiều của mảng a[2][3] tương đương không phần tử mà mỗi phần tử của nó gồm 3 số nguyên nên :
a trỏ tới hàng thứ nhất (a [0][0])
a+1 trỏ tới hàng thứ hai (a[1][0])
- Do đó để duyệt các phần tử của mảng a[2][3] ta dùng con trỏ theo cách sau :
+ (theo *) => ta có công thức a[i][j] = (int*) a + i * n + j
trong đó : int* : con trỏ a (địa chỉ a).
n : số cột.
- float a[2][3] , *p ;
p = (float*)a ; /* chú ý lệnh này */
khi đó : p trỏ tới a[0][0] /* p = & a[0][0] */

p + 1 trỏ tới a[0][1] /* *(p+1) = a[0][1] */
P + 2 trỏ tới a[0][2]
....
p + 5 trỏ tới a[1][2] /* *(p+5) = a[1][2] */
* Tổng quát : a[i][j] = * (p + i* N + 5); trong đó N : số cột)
Kết luận : Mảng 2 chiều có thể chuyển thành mảng 1 chiều nhờ con trỏ.
* Ví dụ : để nhập một số liệu vào mảng 2 chiều kiểu float a[2][3] ta có thể dùng các cách sau:
+ Cách 1 :

```
#include " stdio.h "
main ( )
  { float a[2][3] , *p ; int i ;
    p = (float*)a ; /* lưu ý lệnh này */
    for ( i = 0 ; i < 2*3 ; ++i)
      scanf ( "%f", (p+i)) ; /* (p_+ i ) là địa chỉ */ ( X )
  }
```

+ Cách 2 : Sửa lệnh (X) như sau : scanf ("%f", (float*)a + 1) ;
+ Cách 3 :

```
#include " stdio.h " #define m 2 #define n 3
main ( )
  { float a[m][n] ; int i , j ; float *p ; p = ( float* )a ;
    for ( i=0 ; i<m ; i++ )
    for ( j=0 ; j<n ; j++ )
    scanf ( "%f" , ( p +i*n + j )
    hoặc lệnh scanf ( " %f" , ( float *)a + i * N + j ));
  }
```

+ Cách 4 : sử dụng biến trung gian :

```
#include " stdio.h"
#define dong 2
#define cot 3
main ( )
  { float a[dong][cot] , tam ; int i , j ;
    for ( i = 0 ; i < dong ; i++ ) ;
    for ( j=0 ; j < cot ; ++j )
      { printf ( "\n a[%d][%d] = " , i , j );
        scanf ( " %f " , &tam ) ;
        a[i][j] = tam ;
         }
```

Bài tập : Sắp xếp mảng 2 chiều theo hàng và toàn bộ mảng
+ Mảng con trỏ : là mảng mà mỗi phần tử của nó có thể chứa một địa chỉ nào đó.
Khai báo : < kiểu dữ liệu > < tên mảng > [<chỉ số>].
* Ví dụ : int *a[5] ;
- trong đó : a là mảng gồm 5 ô nhớ liên tiếp, mỗi ô nhớ là 1 biến con trỏ trỏ đến kiểu int ; bản thân a không thể dùng để lưu trữ số liệu.
- Giả sử : a <100> <102> <104> <106> <108> < 110>
a[0] a[1] a[2] a[3] a[4] a[5]
Địa chỉ < 30> < 20> < 10 > < 80 > < 70 > < 100>
7 8 9 10 11
<10> <12> <14>
1 2 3 4 5
<20> <22> <24> <26> <28>
6 12 13
<30> <32> <34>

- a= &a[0] => a = <100> (địa chỉ 100).
- a[0] = < 30 > (địa chỉ bằng 30 : tại địa chỉ 30 con trỏ a[0] trỏ đến địa chỉ <30 > và giả sử tại địa chỉ < 30 > có giá trị là 6).
=> *a[0] = * (<30>> = 6 .
a[1] = < 20 > => *a[1] = 1
a [2] = < 10> => *a[2] = 7 .
Chú ý 1: Xem a là con trỏ 2 lần (con trỏ của con trỏ) :
- a = <100 > => *a = <30 > (do a = &a[0])
=> **a = 6 (do *(<30>)).
- *(*(a + 1) + 2)
*(102)
* (<20> + 2) => *<24> = 3
Chú ý 2 : - int a[5] => a là con trỏ hằng không thay dổi địa chỉ của nó được (nên a++ sai)
- int *a[5] ; => a laf con trỏ động nên thay đổi giá trị được (a++ đúng).
Ví dụ : int *a[5]
For (i = 0 ; i < 5 ; i++)
{ printf ("%d", *a[0]);
a[0]++ ;
}
* Chú ý 3 : mảng 2 chiều chẳng qua là 1 con trỏ 2 lần (con trỏ của con trỏ).
Lý do : a[i][k] ; trong đó đặt b = a[i] => b[k] = a[i][k] ;
+ Công thức : (a[i] = *(a+i)) => (b[i] = *(b+i)).
b[k] = *(b+k)).
b[k] = *(a[i] + k)
= * (*(a+i) + j).
=> a[i][k] = *(*(a+i) + k) ; trong đó *(*(a+i) là con trỏ 2 lần.

5.2. Dữ liệu kiểu xâu ký tự :

- Xâu ký tự : là dãy ký tự đặt trong ngoặc kép . Ví dụ : " Lớp học ". Xâu này được chứa trong 1 mảng kiểu char.
L O P H O C \0
Địa chỉ :<100> <101> <102> NULL : kết thúc chuỗi
=> char *lop ;
lop = " Lop Hoc " ; Đúng : gán địa chỉ của chuỗi cho con trỏ lớp.
+ puts (" Lop Hoc ") ; và puts (lop) đều hiển thị dòng chữ Lop Hoc.
Ví dụ : char Tenlop[10] ;
Printf ("\n Tenlop : ") ; gets(Tenlop) ; => (Nhập vào chuỗi " lớp học ")
Còn nếu chúng ta khai báo như sau là sai :
Char *lop , tenlop [10] ;
Tenlop = " lớp học " ; sai vì Tenlop và chuỗi là 2 con trỏ hằng , không được gán cho nhau .
Muốn gán ta dùng hàm strcpy (Tenlop , "lớp học ");
+ Con trỏ và việc định vị bộ nhớ động :
- Ví dụ 1 :

```
#define N=10 ;
main ( )
  { int a[N] ; int m :
    printf ( " nhập số phần tử m = "); scanf("%d", &m) ;
    for ( i= 0 ; i < m ; i++ )
    scanf ( "%d", &a[i] );
```

- Nhận xét Ví dụ 1 trên : + Nếu m <=N (N =10) : thì sẽ bị dư 1 số biến mảng là (n - m).
+ Nếu m > N (tức là m > 10) : thì chương trình sẽ chạy sai vì ta không đủ biến mảng.
=> Do đó ta phải khắc phục bằng cách : định vị bộ nhớ động. (Bằng hàm malloc và calloc).
* Ví dụ 2 :

```
#include < stdio.h>
#include<alloc.h> hoặc #include <stdio.h >
main ( )
  { int m , *a ;
    printf (" Nhập số phần tử m = " ); scanf ( "%d", &m );
/* Cấp phát và định vị bộ nhớ động */
    a = ( int*) malloc ( m* size of ( int ) ); (1)
      if ( a!= NULL ) /* cấp phát thành công */
      for ( i=0 ; i < m ; i++)
         scanf ( "%d", &a[i] );
         free (a) ; /* giải phóng vùng nhớ mảng */
  }
```

- Hàm malloc ( ) nằm trong thư viện <alloc.h> . Hàm này cung cấp số lượng byte liên tiếp từ phần bộ nhớ còn chưa sử dụng trên máy tính.

+ Ví dụ : malloc (num) = num byte và trả về con trỏ kiểu void trỏ đến địa chỉ bắt đầu của ô nhớ.

- Size of (int) : là số byte mà một biến kiểu int yêu cầu (giá trị = 2)
- (int*) : ép kiểu (type - casing) : coi địa chỉ bắt đầu là int (do malloc trỏ về con trỏ kiểu void , đặc biệt không có kiểu) , có thể nhận bất kỳ địa chỉ kiểu nào (nhờ ép kiểu).
- Muốn sử dụng hàm calloc thay cho hàm malloc => khai báo :

```
a = (int*) calloc ( n, size of (int));
```

* Chú ý : Luôn gán một địa chỉ cho một con trỏ trước khi sử dụng tới nó. Nếu không biến con trỏ sẽ mang một giá trị ngẫu nhiên có thể phá huỷ chương trình.

* Cấp phát bộ nhớ động cho mảng 2 chiều m x n phần tử, m , n nhập từ bàn phím:

+ Ví dụ :

```
#include <stdio.h>
#include <alloc.h>
void main ( )
    { int **a , m, n, OK ;
      printf ( " nhập m = " ); scanf ("%d", &m);
      printf (nhập m = n) ; scanf ( "%d", &n );
      a = ( int** ) malloc ( m*seze of (int *));
      if (a!=NULL ) /*Cấp phát thành công */
        { OK = 1 ;
          for ( i=0 ; i < m ; i++ ) } /* giá trị ban đầu cho biến con trỏ*/
          a[i] = (int*) break ;
          for ( i=0 ; i <m ; i ++ )
             { if !(OK) break ;
               a[i] = (int*) malloc ( n * size of (int));
               if ( a[i] = NULL ) OK = 0 ;
             }
             if(OK)
             { sử dụng a[0][0] , a[0][1]....., a[i][j] ...., a[m][n] }
/* giải phóng vùng nhớ cấp phát */
          if ( a!=NULL )
            { for ( i = 0 ; i < m ; i++)
               if ( a[i] ! = NULL , free ( a[i]);
                   free (a);
            }
       }
```

* Chú ý : ta xem mảng 2 chiều là mảng 1 chiều nên có thể khai báo :

```
a = (int*) malloc ( m*n * size of ( int ));
```

VÀ A[I][J] = A[I*N + J]

Bài tập :
1/ Làm lại các bài tập phần mảng nhưng dùng con trỏ .
2/ Dùng hàm malloc hay calloc nhập mảng n phần tử , sau đó tính tổng các phần tử và sắp xếp mảng giảm dần.
3/ Dùng hàm malloc hay calloc nhập ma trận m x n , sau đó tính tổng và sắp xếp theo tăng dần
+ Mối liên hệ giữa con trỏ và các khái niệm quan trọng:
a/ Con trỏ và hàm :
- Chú ý 1 : bản thân tham số truyền cho hàm không bao giờ bị thay đổi. Nhưng nếu tham số là con trỏ thì giá trị của nó không thay đổi nhưng nội dung được chứa ở địa chỉ đó lại có thể thay đổi.
- Chú ý 2 : Truyền cho hàm một tham số hình thức được khai báo là con trỏ, và khi gọi hàm truyền cho nó một giá trị địa chỉ của biến muốn thay đổi.
- Ví dụ :giả sử tân xây dựng một hàm dùng để hoán vị biến thực, ta viết như sau :
Cách 1 :

```
#include<stdio.h>
void swap (float x , float y ) /* cách 1 sai */
{ float temp ;
  temp = x ; s<y ; y = temp;
}
main ( )
   { float a, b ; a = 10.0 ; b = 20.0 ;
      printf (" khi chưa hoán vị a = %4.0f; b = %4.0f \n" , a , b ) ;
      swap ( a , b ) ;
      printf ( " sau khi hoán vị a = %4.0f ; b = %4.0f \n" , a, b ) ;
```

- Phân tích cái sai của cách 1 của ví dụ trên :
+ Do a, b thuộc hàm main ( ). Khi khai báo sẽ dùng 2 khoảng nhớ (mỗi khoảng 3 byte) . a, b trong lời gọi hàm swap(a,b) là 2 tham số thực.
+ Các đối x, y và biến cục bộ temp được cung cấp khoảng nhớ nhưng địa chỉ khác. Do đó xx, y chỉ tồn tại ở hàm swap(_), còn a, b tồn tại suốt cả quá trình của chương trình nên hàm swap () không làm thay đổi (tức hoán vị) được giá trị của a và b => hàm viết theo cách 1 không đạt yêu cầu => yêu cầu viết lại theo cách 2.
* Cách 2 :

```
void swap (float *x , float *y) /* viết đúng*/
{ float temp ;
temp = *x ; *x = *y ; * y = temp ;
}
main ( )
```

b/ Số học con trỏ (có thể thao tác số học trên nội dung con trỏ)
* Ví dụ :

```
#include < stdio.h>
#include <alloc.h>
main ( )
{ #define N 3
   int *list , i ;
   list = int*) calloc ( N, size of(int));
   *list = 15 ;
   * (list + 1) = 20 ;
   *(list + 2 ) = 30 ;
     printf ( " các địa chỉ là : ");
    for ( i=o ; i < N ; i++)
    printf ("%4d",(list + i));
    printf ("\n chứa các giá trị là : ");
    for ( i=0 ; i < N ; i++)
    printf("%4d", *(list + i));
```

```
     printf("\n");
```
=> list trỏ tới một dãi bộ nhớ dài 6 byte (3*2) có các giá trị là 5,20, 30 . giá trị địa chỉ đầu là 06A => kết quả các địa chỉ là : 06A 06AC 06AE chứa các giá trị là : 5 20 30

c/ Con trỏ và mảng :

- Ví dụ 2 :

```
#include
main ( )
{ #define N 3
  int list [N] , i ;
  list [0] = 5 ; list [1] = 20 ; list[2]=30;
  printf ( " Các địa chỉ là : ");
  for ( i = 0 ; i < N ; i++)
  printf ( "%4p ", &list[i] );
  printf("\n chứa các giá trị là : ");
  for ( i=0; i<N ; i++)
   printf ( "%4d", list [i] ));
}
```

-Kết quả chương trình :

+ Các địa chỉ là : 163A 163C 163E

+ Chứa các giá trị là : 5 20 30

- So với ví dụ 1 thì điều khác duy nhất là giá trị địa chỉ thay đổi. Như vậy ta có thể sử dụng tên của một mảng như con trỏ và ngược lại.

=>{ list + i) = = &(list[i]) và *(list + i) = = list[i]}

d/ Con trỏ và cấu trúc :

- Ta có thể khai báo con trỏ như một biến cấu trúc, cũng như con trỏ của bấu kỳ kiểu dữ liệu nào khác. Điều này cho phép tạo một danh sách móc nối các phần tử (sẽ trình bày chương sau).

e/ Con trỏ tới hàm : dùng để chứa địa chỉ của hàm. Nên kiểu của hàm và con trỏ phải giống nhau.

Ví dụ : #include <stdio.h>

```
Double fmax ( double x, double y ) /* hàm tính max của 2 số */
{ return ( x>y ? x:y ) ; }
/* khai báo và gán tên hàm cho con trỏ hàm */
double (*pf) (double , double ) = fmax ;
main ( )
{ printf ( " In max = % f " , pf(15.5, 20.5 ));
}
```

5.3. Dữ liệu kiểu cấu trúc

- Khái niệm : Cấu trúc là một kiểu dữ liệu kiểu bản ghi(record) , cho phép nhiều loại dữ liệu được nhóm lại với nhau. (Khái niệm cấu trúc trong C tương tự như pascal hay Foxpro).

5.3.1. Khai báo kiểu cấu trúc :

a/ struct tên _ kiểu cấu trúc

```
{
khai báo các thành phần của nó ( các field và kiểu dữ liệu của field)
} < danh sách biến>;
```

- Ví dụ 1 : struct kieu HV ò-> tên kiểu cấu trúc.

```
{ char Ten[30] ;
int namsinh ;float diemTB ;
} HV ; ( biến HV)
```

- Ví dụ 2 : struct kieu HV

{
các thành phần
}
struct kieu HV HV ; /* khai báo biến theo cách 2 */
b/ Dùng toán tử typedef để khai báo kiểu cấu trúc (định nghĩa kiểu mới) ;
- Ví dụ 3 : typedef struct
{ char Ten[30]
  int namsinh ;
  float diemTB ;
} kieu HV ;
kieu HV Hoc vien ;
kieu HV DSLop[20];
kieu HV Lop[ ] = { { "nguyễn văn Đông", 1980, 10.0},
{ " Trần văn Tây", 1982, 5.5},
{ " Phạm văn Nam ", 1979, 6.5}
};
- Ví dụ 4 : struct ngay{
                int ngay ;
              char Thang[10];
                int nam ;
              } ;
type struct
{ char Ten[30] ;
    ngay namsinh ; /* thành phần cấu trúc có kiểu cấu trúc*/
    float diemTB;
} kieu HV ; kieu HV HV;
* Chú ý :
- Khai báo struct phải nằm ở vị trí toàn cục của chương trình, thường sau các #include.
- Cấu trúc thường dùng để xây dựng một bảng các cấu trúc.
+ Ví dụ : kieu HV DSLop[30] ; struct kieu HV person[50];
- Có thể truyền cấu trúc như một tham số hình thức, nhưng với những cấu trúc kích thước lớn sẽ không tối ưu về thời gian lẫn độ nhớ. Khi không nên sử dụng con trỏ cấu trúc.
+ Ví dụ : struc kieu HV *HV ;

5.3.2. Truy cập đến các thành phần của kiểu cấu trúc :

Tên cấu trúc. Tên thành phần
Hoặc Tên cấu trúc. Tên cấu trúc con. Tên thành phần.
- Ví dụ : + nhập vào tên, năm sinh, điểm cho biến cấu trúc học viên (ví dụ 3).
gets(hoc vien.ten) /* nhập " Phạm thị Bắc" và Enter */
scanf("%d ", & hoc vien.namsinh);
scanf("%f", &tam); hoc vien.diem = tam; (*)
+ Nhập năm sinh cho biến học viên ở ví dụ 4 :
scanf("%d",&hv.ngay.namsinh);
* Chú ý : Nếu các thành phần không phải là nguyên(int) => nhập qua trung gian như (*).
puts(hoc vien.ten); => " Phạm thị Bắc"
printf("%d%f", hoc vien.namsinh, hoc vien.diemTB);
* Lệnh gán : + Ta có thể gán 2 biến cấu trúc có cùng kiểu cho nhau :
Ví dụ : hv2=hv1;
+ Gán giá trị đầu cho biến cấu trúc và khai báo một mảng cấu TRÚC(XEM VÍ DỤ 3)
Bài tập : viết chương trình nhập danh sách học viên gồm các trường họ tên, tuổi, điểm, và tìm kiếm trong dánhách có ai tên " Phạm Tèo " không.
Tên Tuổi điểm
HV [0] Nguyễn A 20 5.5

```
HV [1] Trần B 22 6.5
HV [2] Phạm Tèo 25 8.5
HV [3] Lê C 21 7.5
#include <stdio.h>
#define n 10
typedef struct
{ char Ten[30];
  int tuoi ;
  float diem ;
} kieu HV ;
kieu HV HV[11]
void main( )
{ int i ; float tam ; kieu HV HV;
/* nhập dữ liệu cách 1*/
for ( i = 0 ; i < n ; i++)
{ printf ("\n Nhập số liệu cho học viên thứ %d", i ) ;
  printf (" Họ và tên = " ) ; gets ( hv[i].ten);
  printf ("tuổi = "); scanf ( "%d" , &hv[i].tuoi);
  printf("điểm = "); scanf ("%f*c", &tam ); hv[i].diem = tam ;
}
/* cách 2 nhập vào biến cấu trúc và gán hv[i] = h */
for ( i = 0 ; i<n ; i++ )
{ printf("Họ và tên = "); gets(h.ten);
} hv[i] = h ;
/* tìm kiếm Phạm Tèo */
thay = 0 ; i = 0 ; /* thay = 0 : không thấy, thấy = 1 : tìm thấy */
while ((!thay)&&(i <n))
if ( strcmp(hv[i].Ten , " Phạm Tèo ") = = 0 )
{ thay = 1 ;
printf ("%s%d%f ", hv[i].ten , hv[i].tuổi, hv[i].điểm );
}
else i++ ;
if (!thay ) puts ("\n không tìm thấy Phạm Tèo !");
getch( ); }
```

Bài tập : Viết chương trình nhập danh sách gồm na học viên gồm các thông tin như : Họ , tên, điểm pascal , điểm c, sau đó tính điểm trung bình (điemTB) = (diemC*2 + diempascal)/3 .

- Và xét kết quả đậu hay rớt theo qui ước sau :

+ nếu điểm trung bình >= 5 thì kết quả đậu.

+ Nếu điểm trung bình <5 thì kết qua rớt.

+ Nếu điểm trung bình = 4 mà phái = "Nữ" thì kết quả là đậu.

1/ in danh sách vừa nhập gồm họ tên, phái , điểm c, điểm pascal, điểm TB , kết quả .

2/ Sắp xếp giảm dần theo điểm trung bình và in ra.

3/ Nhập vào tên cần tìm và tìm trong danh sách học viên nếu không tìm thấy thì in ra học viên có tên không tìm thấy. Nếu có nhiều học viên có cùng tên cần tìm thì hãy in ra người cuối cùng được tìm thấy.

4/ Giống câu 3 nhưng in ra 2 người tìm thấy đầu tiên (nếu có nhiều người).

5/ Giống câu 3 nhưng in ra người đầu tiên và người cuối cùng (nếu có nhiều ngườ̀i). Nên viết theo từng hàm.

5.3.3. Con trỏ trỏ đến cấu trúc và địa chỉ cấu trúc :

a/ Con trỏ và địa chỉ :

- Ví dụ :

```
typedef struct
{ char Ten[30] ;
```

```
int tuoi ;
float diem ;
} kieu HV ;
```
kieu HV *p , HV , lop[50] ; HS [50] (trong đó : HV là biến cấu trúc, *p : con trỏ cấu trúc dùng để lưu trữ địa chỉ cấu trúc và mảng cấu trúc) (*).
```
main ( )
/* ta có thể gán */
p = &HV ; /* Đúng do (*)*/
p = &lop[i]/*đúng do (*) */
p = lơp ; /* đúng : p = địa chỉ Lop[0] , p = &lop[0] ) do Lop = &Lop[0])
```
b/ truy cập thông qua con trỏ :
- Cách 1 : tên con trỏ -ă tên thành phần.
- Cách 2 : (*tên con trỏ).tên thành phần.
- Ví dụ : p = &HV ; p = &Lop[2] '
=> HV.Ten ĩ p --ă tên;
Lop[2].tuổi ĩ (p*).tuoi ĩ p -ă tuổi ;
```
*p = HV ;
*P = Lop[2]
```
- Giả sử cần nhập số liệu ch vùng trên thì 3 cách viết sau là tương đương :
+ (1) : gets(HV.ten)
+ (2) gets (pă ten) ĩ gets((*p).ten).
+ (3) scanf("%d",&HV.tuoi) ; ĩ scanf("%d", p -ă tuổi);
```
scanf ("%d", (*p).tuoi);
```
- Giả sử cần nhập dữ liệu cho mảng cấu trúc thì các cách viết sau đây tương đương :
+ Ví dụ : p = lop ;
```
for ( i = 0 ; i < n ; i++)
{ gets (lop[i].tên); tương đương với :
. gets((*(lop* i ) ).ten);
.gets(*(p + i ).ten);
.gets ( p[i].ten);
.gets (p ă ten); p++ ;
.gets (*p).ten) ; p++;
```
- Ví dụ : làm lại bài tập mẫu nhưng sử dụng biến con trỏ :
```
#include <stdio.h>
#define n 10
typedef struct
{ char ten[30] ;
int tuoi ;
float diem ;
} kieu HV ;
main ( )
{ kieu HV hv [n], *p , h;
int i ; int thay ; float tam ; int tuổi ; p = hv;
for ( i = 0 ; i < n ; i++)
{ printf (" nhập học viên thứ %d ", i );
  printf("Họ và tên"); gets ( p ă ten);
  printf("tuổi : ") ; scanf ("%d", &tuổi); p ă tuoi = tuoi;
  printf ("diem : ") ; scanf ("%f%*c ", &tam ); p ă diem = tam;
  p++ ; printf ("%c", getchar();
}
/* nhập theo cách 2 qua biến h xong gán *p = h */
/* tìm Phạm Tèo */
```

```
thấy = 0 ; i = 0 ; p = hv ; /* để di chuyển con trỏ về đầu danh sách */
for ( i = 0 ; i < n ; i++ )
if ( strcmp(p ă ten, " Phạm Tèo " ) = = 0 )
{ thấy = 1
   printf ("%s %d%f" , p ă ten, pă tuoi, pă điểm );
   break ;
 else p++ ;
    if (!thay) puts (" không có Phạm Tèo trong danh sách ");
       getch( );
}
```

Bài tập : làm lại bài tập trước nhưng sử dụng con trỏ.

5.3.4. Cấp phát bộ nhớ động cho kiểu dữ liệu cấu trúc :

- Giả sử ta cần quản lý danh sách học viên nên dùng mảng cấu trúc (cấp phát bộ nhớ tĩnh - danh sách đặc) ta phải sử dụng số học viên tối đa => thừa vùng nhớ. Để cấp phát vừa đủ sĩ số học viên như ta muốn => ta dùng phương pháp cấp phát bộ nhớ động hàm malloc hoặc calloc(.)
- Ví dụ : Nhập danh sách n học viên gồm họ tên, điểm và sắp xếp giảm dần theo điểm.

```
#include <stdio.h> #include<conio.h> #include<alloc.h>
#include< string.h>
typedef struct
{ char ten[30] ; int diem ; char kq[5] ; } kieu HV;
kieu HV *lop , *p , tam ;
/* Hàm nhập dan sách */
void nhapDS ( int n , kieu HV lop[ ])
{ int i , diem ;
p = lop ;
for ( i = 0 ; i < n ; i++)
{ printf("nhập Họ tên người thứ %d : " , i +1 ) ; gets ( p ăten);
  printf ( " điểm = " ) ; scanf ( "%d" , &diem ) ; p ă diem = diem ;
  printf ("%c", getchar()); /* khử stdin */
  p++ ;
}
/* Hàm sắp xếp*/
void sapxep ( int n, kieu HV lop[ ])
{ int i , j ; kieu HV tam ;
   for ( i = 0 ; i < n-1 ; i++)
   for ( j=i + 1 ; j< n ; j++)
   if ( lop[i].diem < lop[j].diem )
      { tam = lop[i] ; lop[j] = lop [j] ; lop [j] = tam ; }
/* hàm in danh sách */
void inds( intn, kieu HV lop[ ] )
{ int i ;
   for ( i = 0 ; i < n ; i++ )
  { printf ("%20s%5d ", lop[i].ten,lop[i].diem );
     printf ("\n" ; /* xuống hàng */
/* chương trình chính */
void main ( )
{ int i , j, n , t, diem ;
  printf ("\n Nhập sĩ số : ") ; scanf ( "%d", &n);
 lop = (kieu HV*)malloc ( n * size of ( kieu HV) ) ; printf ("%c", getchar ());
  nhapds (n, lop ) ; sapxep ( n, lop ) ; inds ( in lop );
  getch ( ); }
```

Bài tập :
1/ Làm lại các bài tập phần mảng nhưng dùng con trỏ .
2/ Dùng hàm malloc hay calloc nhập mảng n phần tử , sau đó tính tổng các phần tử và sắp xếp mảng giảm dần.
3/ Dùng hàm malloc hay calloc nhập ma trận m x n , sau đó tính tổng và sắp xếp theo tăng dần

4/ Nhập và xuất 1 danh sách gồm n nhân viên

5/ Nhập, xuất, sắp xếp, tìm kiếm danh sách n sinh viên (sắp xếp theo điểm trung bình, tìm theo tên)

CHƯƠNG 6. DỮ LIỆU KIỂU TỆP

6.1. Khái niệm về tệp tin

Tệp tin hay tệp dữ liệu là một tập hợp các dữ liệu có liên quan với nhau và có cùng một kiểu được nhóm lại với nhau thành một dãy. Chúng thường được chứa trong một thiết bị nhớ ngoài của mấy tính (đĩa mềm, đĩa cứng...) dưới một cái tên nào đó.

Tên tiếng Anh của tệp là **file,** nó được dùng để chỉ ra một hộp đựng các phiếu hay thẻ ghi của thư viện. Một hình ảnh rõ nét giúp ta hình dung ra tệp là tủ phiếu của thư viện. Một hộp có nhiều phiếu giống nhau về hình thức và tổ chức, song lại khác nhau về nội dung. ở đây, tủ phiếu là tệp, các lá phiếu là các thành phần của tệp. Trong máy tính, một đĩa cứng hoặc một đĩa mềm đóng vai trò chiếc tủ (để chứa nhiều tệp).

Tệp được chứa trong bộ nhớ ngoài, điều đó có nghĩa là tệp được lưu trữ để dùng nhiều lần và tồn tại ngay cả khi chương trình kết thúc hoặc mất điện. Chính vì lý do trên, chỉ những dữ liệu nào cần lưu trữ (như hồ sơ chẳng hạn) thì ta nên dùng đến tệp.

Tệp là một kiểu dữ liệu có cấu trúc. Định nghĩa tệp có phần nào giống mảng ở chỗ chúng đều là tập hợp của các phần tử dữ liệu cùng kiểu, song mảng thường có số phần tử cố định, số phần tử của tệp không được xác định trong định nghĩa.

Trong C, các thao tác tệp được thực hiện nhờ các hàm thư viện. Các hàm này được chia làm hai nhóm : nhóm 1 và nhóm 2. Các hàm cấp 1 là các hàm nhập / xuất hệ thống, chúng thực hiện việc đọc ghi như DOS. Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp.

Do các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1 nên trong các chương trình viết trong C, các hàm cấp 2 hay được sử dụng hơn.

6.2. Cấu trúc và phân loại tệp

Một tệp tin dù được xây dựng bằng cách nào đi nữa cũng chỉ đơn giản là một dãy các byte ghi trên đĩa (có giá trị từ 0 đến 255). Số byte của dãy chính là độ dài của tệp.

Có hai kiểu nhập xuất dữ liệu lên tệp : Nhập xuất nhị phân và nhập xuất văn bản.

Nhập xuất nhị phân :

- Dữ liệu ghi lên tệp theo các byte nhị phân như bộ nhớ, trong quá trình nhập xuất, dữ liệu không bị biến đổi.
- Khi đọc tệp, nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (được định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác 0.

Nhập xuất văn bản:

- Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.

- Mã chuyển dòng :

  Khi ghi, một ký tự LF (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF

  Khi đọc, 2 ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF

Mã kết thúc tệp :

Trong khi đọc, nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (bằng -1) và hàm feof(fp) cho giá trị khác 0 (bằng 1).

6.3. Tạo tệp mới để đọc/ghi dữ liệu

Để khai báo sử dụng tệp, ta dùng lệnh sau :

FILE biến_con_trỏ_tệp;

Trong đó biến_con_trỏ_tệp có thể là biến đơn hay một danh sách các biến phân cách nhau bởi dấu phảy (dấu **,**).

Ví dụ :

```
FILE *vb, *np;          /* Khai báo hai biến con trỏ tệp */
```

6.3.2. Mở tệp - hàm fopen :

Cấu trúc ngữ pháp của hàm :

```
FILE *fopen(const char *tên_tệp, const char *kiểu);
```

Nguyên hàm trong : stdio.h .

Trong đó :

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Công dụng :

Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp hai sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm sẽ trả về giá trị NULL.

Bảng sau chỉ ra các giá trị của kiểu :

Tên kiểu	ý nghĩa
"r" "rt"	Mở một tệp để đọc theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w" "wt"	Mở một tệp để ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a" "at"	Mở một tệp để ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"rb"	Mở một tệp để đọc theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có

	lỗi.
"wb"	Mở một tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"ab"	Mở một tệp để ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+" "r+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w+" "w+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+" "a+t"	Mở một tệp để đọc/ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+b"	Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi.
"w+b"	Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+b"	Mở một tệp để đọc/ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.

Chú ý :

Trong các kiểu đọc ghi, ta nên lầm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc ngược lại. Ta sẽ đề cập đến các hàm với tính năng xoá sau này.

Ví dụ :

```
f=fopen("TEPNP","wb");
```

6.3.3. Đóng tệp - hàm fclose :

Cấu trúc ngữ pháp của hàm :

```
int fclose(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

fp là con trỏ ứng với tệp cần đóng.

Công dụng :

Hàm dùng để đóng tệp khi kết thúc các thao tác trên nó. Khi đóng tệp, máy thực hiện các công việc sau :

- Khi đang ghi dữ liệu thì máy sẽ đẩy dữ liệu còn trong vùng đệm lên đĩa
- Khi đang đọc dữ liệu thì máy sẽ xoá vùng đệm
- Giải phóng biến trỏ tệp.
- Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ :

```
fclose(f);
```

6.3.4. Đóng tất cả các tệp đang mở- hàm fcloseall :

Cấu trúc ngữ pháp của hàm :

```
int fcloseall(void);
```

Nguyên hàm trong : stdio.h .

Công dụng :

Hàm dùng để đóng tất cả các tệp đang mở . Nếu lệnh thành công, hàm sẽ cho giá trị bằng số là số tệp được đóng, trái lại nó cho hàm EOF.

Ví dụ : fcloseall();

6.4. Một số hàm xử lý tệp của C

6.4.1. Làm sạch vùng đệm - hàm fflush :

Cấu trúc ngữ pháp của hàm :

```
int fflush(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Công dụng :

Dùng làm sạch vùng đệm của tệp fp. Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ :

```
fflush(f);
```

6.4.2. Làm sạch vùng đệm của các tệp đang mở - hàm fflushall :

Cấu trúc ngữ pháp của hàm :

```
int fflushall(void);
```

Nguyên hàm trong : stdio.h .

Công dụng :

Dùng làm sạch vùng đệm của tất cả các tệp đang mở. Nếu lệnh thành công, hàm sẽ cho giá trị bằng số các tệp đang mở, trái lại nó cho hàm EOF.

Ví dụ :

```
fflushall();
```

6.4.3. Kiểm tra lỗi file - hàm ferror :

Cấu trúc ngữ pháp của hàm :

int ferror(FILE *fp);

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Hàm dùng để kiểm tra lỗi khi thao tác trên tệp fp. Hàm cho giá trị 0 nếu không có lỗi, trái lại hàm cho giá trị khác 0.

6.4.4. Kiểmtra cuối tệp - hàm feof :

Cấu trúc ngữ pháp của hàm :

int feof(FILE *fp);

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.

6.4.5. Truy nhập ngẫu nhiên - các hàm di chuyên con trỏ chỉ vị :

a. Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind :

Cấu trúc ngữ pháp :

void rewind(FILE *fp);

Nguyên hàm trong : stdio.h .

Trong đó fp là con trỏ tệp.

Công dụng :

Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu.

Ví dụ :

rewind(f);

b. Chuyển con trỏ chỉ vị trí cần thiết - Hàm fseek :

Cấu trúc ngữ pháp :

int fseek(FILE *fp, long sb, int xp);

Nguyên hàm trong : stdio.h .

Trong đó

fp là con trỏ tệp.

sb là số byte cần di chuyển.

xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đó.

xp có thể nhận các giá trị sau :

xp=SEEK_SET hay 0 : Xuất phát từ đầu tệp.

xp=SEEK_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

xp=SEEK_END hay 2 : Xuất phát từ cuối tệp.

Công dụng :

Chuyển con trỏ chỉ vị của tệp fp về vị trí xác định bởi xp qua một số byte xác định bằng giá trị tuyệt đối của sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại nó sẽ di chuyển về đầu tệp. Khi thành công, hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

Chú ý :

Không nên dùng fseek trên tệp tin văn bản, do sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

Ví dụ :

```
fseek(stream, SEEK_SET, 0);
```

c. Vị trí hiện tại của con trỏ chỉ vị - Hàm ftell :

Cấu trúc ngữ pháp :

```
int ftell(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó

fp là con trỏ tệp.

Công dụng :

Hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp **fp**) khi thành công. Số thứ tự tính từ 0. Trái lại hàm cho giá trị -1L.

Ví dụ :

Sau lệnh fseek(fp,0,SEEK_END);

ftell(fp) cho giá trị 3.

Sau lệnh fseek(fp,-1,SEEK_END);

ftell(fp) cho giá trị 2.

6.4.6. Ghi các mẫu tin lên tệp - hàm fwrite :

Cấu trúc ngữ pháp của hàm :

```
int fwrite(void *ptr, int size, int n, FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

size là kích thước của mẫu tin theo byte

n là số mẫu tin cần ghi

fp là con trỏ tệp

Công dụng :

Hàm ghi n mẫu tin kích thước **size** byte từ vùng nhớ **ptr** lên tệp **fp**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự ghi được.

Ví dụ :

```
#include "stdio.h"
struct mystruct
{
 int i;
 char ch;
};
main()
{
  FILE *stream;
  struct mystruct s;
  stream = fopen("TEST.TXT", "wb") /* Mở tệp TEST.TXT */
  s.i = 0;
  s.ch = 'A';
  fwrite(&s, sizeof(s), 1, stream); /* Viết cấu trúc vào tệp */
  fclose(stream); /* Đóng tệp */
  return 0;
}
```

6.4.7. Đọc các mẫu tin từ tệp - hàm fread :

Cấu trúc ngữ pháp của hàm :

```
int fread(void *ptr, int size, int n, FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

size là kích thước của mẫu tin theo byte

n là số mẫu tin cần ghi

fp là con trỏ tệp

Công dụng :

Hàm đọc n mẫu tin kích thước **size** byte từ tệp **fp** lên lên vùng nhớ **ptr**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự đọc được.

Ví dụ :

```
#include "string.h"
#include "stdio.h"
main()
{
  FILE *stream;
  char msg[] = "Kiểm tra";
  char buf[20];
 stream = fopen("DUMMY.FIL", "w+");
  /* Viết vài dữ liệu lên tệp */
  fwrite(msg, strlen(msg)+1, 1, stream);
 /* Tìm điểm đầu của file */
  fseek(stream, SEEK_SET, 0);
  /* Đọc số liệu và hiển thị */
  fread(buf, strlen(msg)+1, 1, stream);
  printf("%s\n", buf);
  fclose(stream);
  return 0;
}
```

6.4.8. Nhập xuất ký tự :

a. Các hàm putc và fputc :

Cấu trúc ngữ pháp :

```
int putc(int ch, FILE *fp);
int fputc(int ch, FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

ch là một giá trị nguyên

fp là một con trỏ tệp.

Công dụng :

Hàm ghi lên tệp fp một ký tự có mã bằng

m=ch % 256.

ch được xem là một giá trị nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại cho EOF

Ví dụ :

```
#include "stdio.h"
main()
{
  char msg[] = "Hello world\n";
  int i = 0;
  while (msg[i])
    putc(msg[i++], stdout);  /* stdout thiết bị ra chuẩn - Màn hình*/
  return 0;
}
```

b. Các hàm getc và fgettc :

Cấu trúc ngữ pháp :

```
int gretc(FILE *fp);
int fputc(FILE *fp);
```

Nguyên hàm trong : stdio.h .

Trong đó :

fp là một con trỏ tệp.

Công dụng :

Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm sẽ cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm sẽ trả về EOF.

Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10. Khi gặp mã 26 hàm sẽ trả về EOF.

Ví dụ :

```
#include "string.h"
#include "stdio.h"
#include "conio.h"
main()
{
  FILE *stream;
  char string[] = "Kiem tra";
  char ch;
  /* Mở tệp để cập nhật*/
  stream = fopen("DUMMY.FIL", "w+");
  /*Viết một xâu ký tự vào tệp */
  fwrite(string, strlen(string), 1, stream);
  /* Tìm vị trí đầu của tệp */
```

```
  fseek(stream, 0, SEEK_SET);
  do
  {
    /* Đọc một ký tự từ tệp */
    ch = fgetc(stream);
    /* Hiển thị ký tự */
    putch(ch);
  } while (ch != EOF);
  fclose(stream);
  return 0;
}
```

6.2.9. Xoá tệp - hàm unlink:

Cấu trúc ngữ pháp :

int unlink(const char *tên_tệp)

Nguyên hàm trong : dos.h, io.h, stdio.h .

Trong đó

tên_tệp là tên của tệp cần xoá.

Công dụng :

Dùng để xoá một tệp trên đĩa. Nếu thành công, hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

Ví dụ :

```
#include <stdio.h>
#include <io.h>
int main(void)
{
  FILE *fp = fopen("junk.jnk","w");
  int status;
  fprintf(fp,"junk");
  status = access("junk.jnk",0);
  if (status == 0)
    printf("Tệp tồn tại\n");
  else
    printf("Tệp không tồn tại\n");
  fclose(fp);
  unlink("junk.jnk");
```

```
  status = access("junk.jnk",0);
  if (status == 0)
    printf("Tệp tồn tại\n");
  else
    printf("Tệp không tồn tại\n");
  return 0;
}
```

6.5. Bài tập áp dụng

Xây dựng các chương trình đã cho trong chương 5 sao cho dữ liệu được ghi và file.

CHƯƠNG 7. ĐỒ HOẠ

Chương này sẽ giới thiệu các hàm và thủ tục để khởi động hệ đồ hoạ, vẽ các đường và hình cơ bản như hình tròn, cung elip, hình quạt, đường gãy khúc, đa giác, đường thẳng, hình chữ nhật, hình hộp chữ nhật....

Các hàm và thủ tục đồ hoạ được khai báo trong file graphics.h.

7.1. Giới thiệu chung

Mục đích của việc khởi động hệ thống đồ hoạ là xác định thiết bị đồ hoạ (màn hình) và mode đồ hoạ sẽ sử dụng trong chương trình. Để làm công việc này, ta có hàm sau :

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 hoạ.
- graphdriver cho biết màn hình đồ hoạ sử dụng trong chương trình.
- graphmode cho biết mode đồ hoạ sử dụng trong chương trình.

Bảng dưới đây cho các giá trị khả dĩ của graphdriver và graphmode :

graphdriver	**graphmode**	**Độ phân giải**
DETECT (0)		
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)	640x480
EGA (3)	EGAL0 (0)	640x200
	EGAHi (1)	640x350
EGA64 (4)	EGA64LO (0)	640x200
	EGA64Hi (1)	640x350
EGAMONO (5)	EGAMONOHi (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 mầu
	PC3270HI (1)	1024x768 256 mầu

Chú ý :

- Bảng trên cho ta các hằng và giá trị của chúng mà các biến graphdtriver và graphmode có thể nhận. Chẳng hạn hằng DETECT có giá trị 0, hằng VGA có giá trị 9, hằng VGALO có giá trị 0 vv...

  Khi lập trình ta có thể thay thế vào vị trí tương ứng của chúng trong hàm tên hằng hoặc giá trị của hằng đó.

Ví dụ :

Giả sử máy tính có màn hình VGA, các tập tin đồ hoạ chứa trong thư mục C:\TC \BGI, khi đó ta khởi động hệ thống đồ hoạ như sau :

```
#include "graphics.h"
main()
        {
                int mh=VGA,mode=VGAHI; /*Hoặc mh=9,mode=2*/
                initgraph(&mh,&mode,"C:\\TC\\BGI");
                /* Vì kí tự \ trong C là kí tự đặc biệt nên ta phải gấp đôi nó */
}
```

- Bảng trên còn cho thấy độ phân giải còn phụ thuộc cả vào 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. Với màn hình EGA trên : 639, Hàm getmaxy() cho giá trị cực đại của số điểm theo chiều dọc của màn hình. Với màn hình EGA trên : 199).
- 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 hoạ ở độ phân giải cao nhất ứng với màn hành đang sử dụng cũng được phát hiện và trị số của nó được gán cho biến graphmode.

Như vậy dùng hằng số DETECT chẳng những có thể khởi động được hệ thống đồ hoạ với 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 :

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");
        printf("\n Gia tri so cua man hinh la : %d",mh);
        printf("\n Gia tri so mode do hoa la : %d",mode);
        closegraph();
    }
```

- Nếu chuỗi dùng để xác định driverpath là chuỗi rỗng thì chương trình dịch sẽ tìm kiếm các file điều khiển đồ hoạ trên thư mục chủ (Thư mục hiện thời).

7.2. Các hàm đặt màu, vẽ điểm, tô màu

7.2.1. Mẫu và màu :

- **Đặt màu nền :**

  Để đặt màu cho nền ta dùng thủ tục sau :

  void setbkcolor(int màu);

- **Đặt màu đường vẽ :**

  Để đặt màu vẽ đường ta dùng thủ tục sau :

  void setcolor(int màu);

- **Đặt mẫu (kiểu) tô và màu tô :**

  Để đặt mẫu (kiểu) tô và màu tô ta dùng thủ tục sau :

  void setfillstyle(int mẫu, int màu);

Trong cả ba trường hợp **màu** xác định mã của màu.

Các giá trị khả dĩ của **màu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của 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
LIGHTMAGENTA	13	Tím nhạt
YELLOW	14	Vàng
WHITE	16	Trắng

Các giá trị khả dĩ của **mẫu** cho bởi bảng dưới đây :

Bảng các giá trị khả dĩ của mẫu

Tên hằng	Giá trị số	Kiểu mẫu tô
EMPTY_FILL	0	Tô bằng mầu 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_FILL	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 giải màu :

Để thay đổi giải màu đã được định nghĩa trong bảng trên, ta sử dụng hàm :

void setpalete(int số_thứ_tự_màu, int màu);

Ví dụ :

Câu lệnh :

setpalete(0,lightcyan);

biến màu đầu tiên trong bảng màu thành màu xanh lơ nhạt. Các màu khác không bị ảnh hưởng.

- **Lấy giải màu hiện thời :**

+ Hàm getcolor trả về mầu đã xác định bằng thủ tục setcolor ngay trước nó.

+ Hàm getbkcolor trả về mầu đã xác định bằng hàm setbkcolor ngay trước nó.

7.2.2. Vẽ và tô màu :

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.

7.2.2.1. 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 ta dùng hàm :

void arc(int x, int y, int gd, int gc, int r);

Trong đó :

(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 ta dùng hàm :

void circle(int x, int y, int r);

Trong đó :

(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 ta dùng hàm :

```
void ellipse(int x, int y, int gd, int gc, int xr, int yr);
```

Trong đó :

(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 ta dùng hàm :

```
void pieslice(int x, int y, int gd, int gc, int r);
```

Trong đó :

(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();
}
```

7.2.3. Vẽ đường gấp khúc và đa giác :

- **Vẽ đường gấp khúc :**

Muốn vẽ đường gấp khúc đi qua n điểm : (x1,y1), (x2,y2), ..., (xn,yn) thì trước hết ta phải gán các toạ độ (xi,yi) cho một mảng a kiểu int nào đó theo nguyên tắc sau :

Toạ độ x1 gán cho a[0]
Toạ độ y1 gán cho a[1]
Toạ độ x2 gán cho a[2]
Toạ độ y2 gán cho a[3]
....
Toạ độ xn gán cho a[2n-2]
Toạ độ yn gán cho a[2n-1]

Sau đó gọi hàm :

```
drawpoly(n,a);
```

Nếu điểm cuối cùng (xn,yn) trùng với điểm đầu (x1,y1) thì ta nhận được một đường gấp khúc khép kín.

- **Tô màu đa giác :**

Giả sử ta có a là mảng đã đề cập đến trong mục trên, khi đó ta gọi hàm :

```
fillpoly(n,a);
```

sẽ vẽ và tô màu một đa giác có đỉnh là các điểm (x1,y1), (x2,y2), ..., (xn,yn)

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 :**

Để vẽ đường thẳng nối hai điểm bất kỳ có toạ độ (x1,y1) và (x2,y2) ta sử dụng hàm sau :

```
void line(int x1, int y1, int x2, int y2);
```

Con chạy đồ hoạ giữ nguyên vị trí.

Để vẽ đường thẳng nối từ điểm con chạy đồ hoạ đến một điểm bất có toạ độ (x,y) ta sử dụng hàm sau :

```
void lineto(int x, int y);
```

Con chạy sẽ chuyển đến vị trí (x,y).

Để vẽ một đường thẳng từ ví trí con chạy hiện tại (giả sử là điểm x,y) đến điểm có toạ độ (x+dx,y+dy) ta sử dụng hàm sau :

```
void linerel(int dx, int dy);
```

Con chạy sẽ chuyển đến vị trí (x+dx,y+dy).

- **Di chuyển con chạy đồ hoạ :**

Để di chuyển con chạy đến vị trí (x,y), ta sử dụng hàm sau :

```
void moveto(int x, int y);
```

- **Chọn kiểu đường :**

Hàm `void setlinestyle(int kiểu_đường, int mẫu, int độ_dày);`

tác động đến nét vẽ của các thủ tục vẽ đường line, lineto,linerel , circle, rectangle (hàm vẽ hình chữ nhật, ta sẽ học trong phần vẽ miền ở dưới).

Hàm này sẽ cho phép ta xác định ba yếu tố khi vẽ đường thẳng, đó là : Kiểu đường, bề dày và mẫu tự tạo.

Dạng đường do tham số **kiểu_đường**xác định. Bảng dưới đây cho các giá trị khả dĩ của **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

Bề dày của đường vẽ do tham số **độ_dày** xác định,. bảng dưới đây cho các giá trị khả dĩ của **độ_dày** :

Tên hằng	Giá trị số	Bề dày
NORM_WIDTH	1	Bề dày bình thường
THICK_WIDTH	3	Bề dày gấp ba

Mẫu tự tạo : Nếu tham số thứ nhất là USERBIT_LINE thì ta có thể tạo ra mẫu đường thẳng bằng tham số **mẫu.** Ví dụ ta xét đoạn chương trình :

```
int pattern = 0x1010;
setlinestile(USERBIT_LINE,pattern,NORM_WIDTH);
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 1010, trong hệ 2 là :

0001 0000 0001 0000

Bit 1 sẽ cho điểm sáng, bit 0 sẽ làm tắt điểm ảnh.

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 chạy ở vị trí ( 320,100 ) */
        line(20,20,620,20); /* con chạy vẫn ở vị trí ( 320,100 ) */
        linerel(-300,80);
        lineto(620,180);
        lineto(620,20);
        getch();
        closegraph();
}
```

7.2.4. Vẽ điểm, miền :

- **Vẽ điểm :**

  Hàm :

  ```
  void putpixel(int x, int y, int color);
  ```

sẽ tô điểm (x,y) theo mầu xác định bởi **color.**

Hàm :

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu mầu của điểm ảnh ở vị trí (x,y).

Chú ý :

Nếu điểm này chưa được tô màu bởi các hàm vẽ hoặc hàm putpixel (mà chỉ mới được tạo màu nền bởi setbkcolor) thì hàm cho giá trị 0.

- **Tô miền :**

  Để tô màu cho một miền nào đó trên màn hình, ta dùng hàm sau :

  ```
  void floodfill(int x, int y, int border);
  ```

ở đây :

(x,y) là toạ độ của một điểm nào đó gọi là điểm gieo.

Tham số border chứa mã của màu.

Sự hoạt động của hàm floodfill phụ thuộc vào giá trị của x,y,border và trạng thái màn hình.

+ Khi trên màn hình có một đường cong khép kín hoặc đường gấp khúc khép kín mà mã màu của nó bằng giá trị của border thì :

- Nếu điểm gieo (x,y) nằm trong miền này thì miền giới hạn phía trong đường sẽ được tô màu.

- Nếu điểm gieo (x,y) nằm ngoài miền này thì miền phía ngoài đường sẽ được tô màu.

+ Trong trường hợp khi trên màn hình không có đường cong nào như trên thì cả màn hình sẽ được tô màu.

Ví dụ :

Vẽ một đường tròn màu đỏ trên màn hình màu xanh. Toạ độ (x,y) của điểm gieo được nạp từ bàn phím. Tuỳ 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"
main()
    {
        int mh=mode=0, x, y;
        printf("\nVao toa do x,y:");
        scanf("%d%d",&x,&y);
        initgraph(&mh,&mode,"");
        if (graphresult != grOk) exit(1);
        setbkcolor(BLUE);
        setcolor(RED);
        setfillstyle(11,YELLOW);
        circle(320,100,50);
        moveto(1,150);
        floodfill(x,y,RED);
        closegraph();
    }
```

7.3. Các hàm vẽ hình cơ bản

7.3.1. Hình chữ nhật :

- Hàm :

  void rectangle(int x1, int y1, int x2, int y2);

sẽ vẽ một hình chữ nhật có các cạnh song song với các cạnh của màn hình. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm :

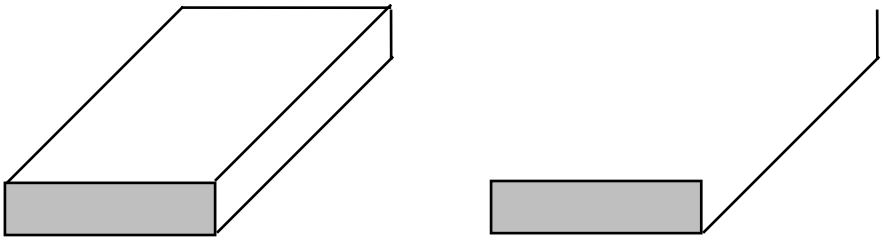
  void bar(int x1, int y1, int x2, int y2);

sẽ vẽ và tô màu một hình chữ nhật. Toạ độ đỉnh trái trên của hình chữ nhật là (x1,y1) và toạ độ đỉnh phải dưới của hành chữ nhật là (x2,y2).

- Hàm :

  void bar3d(int x1, int y1, int x2, int y2, int depth, int top);

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các toạ độ (x1,y1), (x2,y2). Hình chữ nhật này được tô màu thông qua hàm setfillstyle . Tham số **depth** xác định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số **top** có thể nhận các giá trị 1 hay 0 và khối 3 chiều tương ứng sẽ có nắp hoặc không.



top=1 top=0

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=mode=0;
                initgraph(&mh,&mode,"");
                if (graphresult != grOk) exit(1);
                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);
                closegraph();
        }
```

7.3.2. Cửa sổ (Viewport) :

- **Thiết lập viewport :**

Viewport là một vùng chữ nhật trên màn hình đồ hoạ. Để thiết lập viewport ta dùng hàm :

void setviewport(int x1, int y1, int x2, int y2, int clip);

trong đó (x1,y1) là toạ độ góc trên bên trái, (x2,y2) là toạ độ góc dưới bên phải. Bốn giá trị này vì thế phải thoả mãn :

$$0 \le x1 \le x2$$

$$0 \le y1 \le y2$$

Tham số clip có thể nhận một trong hai giá trị :

clip=1 không cho phép vẽ ra ngoài viewport.

clip=0 cho phép vẽ ra ngoài viewport.

Ví dụ :

```
setviewport(100,50,200,150,1);
```

Lập nên một vùng viewport hình chữ nhật có toạ độ góc trái cao là (100,50) và toạ độ góc phải thấp là (200,150) (là toạ độ trước khi đặt viewport).

Chú ý :

Sau khi lập viewport, ta có hệ toạ độ mới mà góc trên bên trái sẽ có toạ độ (0,0).

- **Nhận diện viewport hiện hành :**

Để nhận viewport hiện thời ta dùng hàm :

```
void getviewsetting(struct viewporttype *vp);
```

ở đây kiểu viewporttype đã được định nghĩa như sau :

```
struct viewporttype
    {
        int left,top,right,bottom;
        int clip;
    };
```

- **Xóa viewport :**

Sử dụng hàm :

```
void clearviewport(void);
```

- **Xoá màn hình, đưa con chạy về tạo độ (0,0) của màn hình :**

Sử dụng hàm :

```
void cleardevice(void);
```

- **Toạ độ âm dương :**

Nhờ sử dụng viewport có thể viết các chương trình đồ hoạ theo toạ độ âm dương. Muốn vậy ta thiết lập viewport và cho clip bằng 0 để có thể vẽ ra ngoài giới hạn của viewport. Sau đây là đoạn chương trình thực hiện công việc trên :

```
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 toạ 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 toạ độ âm dương. Hoành độ 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=mode=DETECT;
        int x,y,i;
        initgraph(mh,mode,"");
        if (graphresult!=grOK ) exit(1);
        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()/2,0);
        settextjustify(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();
}
```

7.3.3. Xử lý văn bản trên màn hình đồ hoạ

- **Hiển thị văn bản trên màn hình đồ hoạ :**

Hàm :

```
void outtext(char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí con trỏ đồ hoạ hiện thời.

Hàm :

```
void outtextxy(int x, int y,char *s);
```

cho hiện chuỗi ký tự (do con trỏ s trỏ tới) tại vị trí (x,y).

Ví dụ :

Hai cách viết dưới đây :

```
outtextxy(50,50," Say HELLO");
```

và

```
moveto(50,50);
outtext(" Say HELLO");
```

cho cùng kết quả.

- **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 :

```
void settextstyle(int font, int direction, int charsize);
```

Tham số font để 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

Tham số derection 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.

Tham số 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,VERT_DIR,2);
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.

- **Đạt vị trí hiển thị của các xâu ký tự cho bởi outtext và outtextxy :**

Hàm settextjustify cho phép chỉ định ra nơi hiển thị văn bản của outtext theo quan hệ với vị trí hiện tại của con chạy và của outtextxy theo quan hệ với toạ độ (x,y);

Hàm này có dạng sau :

```
void settextjustify(int horiz, int vert);
```

Tham số horiz có thể là một trong các hằng số sau :

```
LEFT_TEXT=0      ( Văn bản xuất hiện bên phải con chạy).
CENTER_TEXT    ( Chỉnh tâm văn bản theo vị trí con chạy).
RIGHT_TEXT       (Văn bản xuất hiện bên trái con chạy).
```

Tham số vert có thể là một trong các hằng số sau :

```
BOTTOM_TEXT=0 ( Văn bản xuất hiện phía trên con chạy).
CENTER_TEXT=1  ( Chỉnh 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");
```

sẽ cho dòng chữ ABC trong đó điểm (100,100) sẽ nằm dưới chữ B.

- **Bề rộng và chiều cao văn bản :**

Chiều cao :

Hàm :

textheight(char *s);

cho chiều cao (tính bằng pixel) của chuỗi do con trỏ s trỏ tới.

Ví dụ 1 :

Với font bit map và hệ số phóng đại là 1 thì textheight("A") ch giá trị là 8.

Ví dụ 2 :

```
#include "stdio.h"
#include "graphics.h"
main()
	{
		int mh=mode=DETECT, y,size;
		initgraph(mh,mode,"C:\\TC\\BGI");
		y=10;
		settextjustify(0,0);
		for (size=1;size<5;++size)
			{
				settextstyle(0,0,size);
				outtextxy(0,y,"SACRIFICE");
				y+=textheight("SACRIFICE")+10;
			}
		getch();
		closegraph();
	}
```

Bề rộng :

Hàm :

textwidth(char *s);

cho bề rộng chuỗi (tính theo pixel) mà con trỏ s trỏ tới dựa trên chiều dài chuỗi, kích thước font chữ, hệ số phóng đại.

Bài tập:

1. Vẽ và tô màu các hình cơ bản
2. Vẽ hình tròn rồi cho chạy đi chạy lại trên màn hình

Một số đề thi mẫu

Đề 1

Câu 1: Viết chương trình nhập vào một dãy số gồm n phần tử. In các số đã nhập ra màn hình và tính tổng các số chẵn trong dãy.

Câu 2: Viết chương trình nhập vào một danh sách gồm n sinh viên (có các thông tin: họ tên, mã số, điểm trung bình). In danh sách ban đầu và các sinh viên có điểm trung bình >7 ra màn hình.

Đề 2

Câu 1: Nhập vào một ma trận có n hàng, m cột. Tính tổng các phần tử trên đường chéo chính. In kết quả vào file.

Câu 2: Nhập vào một danh sách gồm n nhân viên (có các thông tin: tên nhân viên, hệ số lương, địa chỉ). Tính lương cho các nhân viên theo công thức lương nhân viên=hệ số lương * 730000. In danh sách các nhân viên có lương >5 triệu vào file và ra màn hình.

Đề 3

Câu 1: Nhập vào một ma trận vuông cỡ n. In ra các phần tử lớn nhất của từng hàng trong ma trận.

Câu 2: Nhập vào một danh sách gồm n sinh viên (có các thông tin: họ tên, mã số, điểm trung bình). Sắp xếp danh sách theo họ tên và in kết quả ra màn hình.

Đề 4

Câu 1: Nhập vào một dãy số thực có n phần tử. Tìm phần tử lớn nhất trong dãy và vị trí của nó.

Câu 2: Nhập vào một danh sách gồm n sinh viên (có các thông tin: họ tên, mã số, điểm trung bình). Tìm sinh viên có tên là xyz nào đó, với xyz là tên bất kỳ nhập vào từ bàn phím.

Đề 5

Câu 1: Tìm tất cả các số nguyên tố trong đoạn [n, m]. Ghi kết quả tìm được vào một file.

Câu 2: Nhập vào một danh sách gồm n nhân viên (có các thông tin: tên nhân viên, hệ số lương, địa chỉ). Tính lương cho các nhân viên theo công thức lương nhân viên=hệ số lương * 730000. In danh sách các nhân viên có lương một năm >50 triệu vào file và ra màn hình.

TÀI LIỆU THAM KHẢO

1. Các tài liệu tiếng Việt :

1.1. Ngô Trung Việt - Ngôn ngữ lập trình C và C++ - Bài giảng- Bài tập - Lời giải mẫu
NXB giao thông vận tải 1995

1.2. Viện tin học - Ngôn ngữ lập trình C
Hà nội 1990

1.3. Lê Văn Doanh - 101 thuật toán và chương trình bằng ngôn ngữ C

2. Các tài liệu tiếng Anh :

2.1. B. Kernighan and D. Ritchie - The C programming language
Prentice Hall 1989

2.2. Programmer's guide Borland C++ Version 4.0
Borland International, Inc 1993

2.3. Bile - Nabaiyoti - TURBO C++
The Waite Group's UNIX 1991