Chương 1. Cấu trúc chung của chương trình C

I. Giới thiệu về ngôn ngữ C

II. Các phần tử cơ bản của ngôn ngữ C

III. Cấu trúc chung của một chương trình C (viết trên DOS)

IV. Các bước viết và chạy thử chương trình C

I. Giới thiệu về ngôn ngữ C

1. Ngôn ngữ lập trình C
2. Trình biên dịch C

1. Ngôn ngữ lập trình C

- Năm 1973 ngôn ngữ lập trình C ra đời với mục đích ban đầu là để viết hệ điều hành Unix trên máy tính mini PDP. Sau đó C đã được sử dụng rộng rãi trên nhiều loại máy tính khác nhau và đã trở thành một ngôn ngữ lập trình có cấu trúc rất được ưa chuộng.
- C là ngôn ngữ lập trình bậc trung, có tính cấu trúc và định kiểu.

2. Trình biên dịch C

- C++ là một ngôn ngữ lập trình phát triển từ ngôn ngữ C, vì vậy trình biên dịch C++ cũng biên dịch được các chương trình viết bằng ngôn ngữ C.
- Borland C++ 3.1 là một chương trình biên dịch các chương trình C++ và C viết trên DOS và cả trên Windows.
- Là phần mềm của hãng Borland (Mỹ).
- Việc sử dụng Borland C++ 3.1 trên DOS giống như Turbo Pascal 7.0. Tất cả các thao tác mở, đóng tệp, soạn thảo chương trình, biên dịch và chạy thử chương trình giống như Turbo Pascal.

II. Các phần tử cơ bản của ngôn ngữ C

1. Bộ ký tự
2. Từ khoá
3. Các tên tự đặt
4. Các tên chuẩn
5. Dấu chấm phẩy
6. Lời chú thích

1. Bộ ký tự của ngôn ngữ C

- Mọi ngôn ngữ lập trình đều được xây dựng trên một bộ ký tự nào đó. Các ký tự ghép lại với nhau tạo thành các từ. Các từ lại được liết kết với nhau theo một quy tắc nào đó để tạo thành các câu lệnh. Một chương trình bao gồm nhiều câu lệnh diễn đạt một thuật toán để giải một bài toán nào đó.
- Bộ ký tự của ngôn ngữ C gồm có các ký tự sau:
  - 26 chữ cái hoa: A, B,C,…Z và 26 chữ cái thường: a…z
  - 10 chữ số: 0, 1, 2,…, 9
  - Các ký hiệu toán học: + - * / =) (

1. Bộ ký tự của ngôn ngữ C

- Ký tự gạch nối _
- Các dấu chấm câu và các ký tự đặc biệt khác: . , ; : [] ? ! \ & | % # $ ….
- Dấu cách là một khoảng trống dùng để ngăn cách giữa các từ.

Chú ý: Khi viết chương trình ta không được sử dụng các ký tự không có trong tập ký tự trên.

2. Từ khoá

- Từ khoá là những từ của riêng ngôn ngữ C. Chúng được sử dụng cho các kiểu dữ liệu, toán tử và câu lệnh.
- Các từ khoá của C gồm có:

asm	_asm	__asm	auto	break	case
cdecl	_cdecl	__cdecl	char	class	const
continue	_cs	__cs	default	delete	do
double	_ds	__ds	else	enum	_es
__es	_export	__export	extern	far	_far

2. Từ khoá

__far _fastcall __fastcall float for friend
goto huge _huge __huge if inline
int interrupt _interrupt __interrupt _loadds __loadds
long near _near __near new operator
pascal _pascal __pascal private protected public
register return _saveregs __saveregs _seg __seg
short signed sizeof _ss __ss static
struct switch template this typedef union
unsigned virtual void volatile while

3. Các tên tự đặt

- Tên dùng để xác định các đại lượng khác nhau trong chương trình như tên hằng, tên biến, tên hàm, tên con trỏ, tên cấu trúc, tên tệp, tên nhãn,…
- Tên là một dãy ký tự có thể là chữ cái, chữ số hoặc dấu gạch nối song ký tự đầu tiên phải là chữ cái hoặc dấu gạch nối. Tên không được đặt trùng với từ khoá.
- Một số ví dụ về tên đặt sai:

  3XYZ_7 R#3
  F(x) case
  Al pha

4. Tên chuẩn

- Tên chuẩn là các tên đã được trình biên dịch đặt. Tên chuẩn có thể là tên hằng, tên các hàm.

Ghi nhớ: + Các từ khoá, tên tự đặt, tên chuẩn phân biệt chữ hoa chữ thường, nghĩa là viết hoa, viết thường là khác nhau.

Ví dụ: Tên AB khác với tên ab

+ Riêng từ khoá, tên chuẩn luôn luôn dùng chữ thường, tên chuẩn là hằng thường là chữ hoa.

5. Dấu chấm phẩy

- Dấu chấm được dùng để ngăn cách giữa các câu lệnh. Dấu chấm phẩy thường đặt ở cuối câu lệnh và không thể thiếu được trong chương trình C.

Ví dụ:

```
float x;
x = 10.5;
x = 2*x – 2.5;
```

6. Lời giải thích

- Lời giải thích do người lập trình đưa vào để cho chương trình dễ hiểu, dễ đọc. Lời giải thích có thể đặt bất kỳ đâu trong chương trình nhưng phải đặt trong cặp

  /* */

  hoặc đặt sau //
- Dùng /* và */ khi lời giải thích nằm trên nhiều dòng, dùng // khi lời giải thích nằm trên một dòng.

III. Cấu trúc chung của một chương trình C (viết trên DOS)

```
//Khai báo sử dụng thư viện chương trình con, thư viện lớp
#include<stdio.h>          <-- Tương đương với USES trong PASCAL
   …….
//Khai báo các hàm (chương trình con)
   …….
int main()                 <-- Tương đương với BEGIN trong PASCAL
{
   //Khai báo các biến, hằng
   …….
   //Các lệnh của chương trình
   ……..
   return 0;               <-- Tương đương với END trong PASCAL
 }
//Định nghĩa các hàm
   ……..
```

Thân chương trình chính

Các bước viết và chạy thử chương trình C

Bước 1: Soạn thảo chương trình

- Chạy trình soạn thảo văn bản text
- Gõ vào chương trình và ghi thành tệp có đuôi .c

Bước 2: Biên dịch chương trình

- Chạy trình biên dịch C/C++ để biên dịch chương trình
- Nếu có lỗi về mặt cú pháp thì quay lại bước 1 để sửa.

Bước 3: Chạy thử chương trình

- Từ trong IDE, ấn Ctrl+F9
- Nhập vào dữ liệu mẫu, nếu thấy kết quả sai thì kiểm tra lại thuật giải rồi quay lại bước 1 viết lại chương trình.

Chương 2. Các kiểu dữ liệu cơ sở

I. Khái niệm về kiểu dữ liệu
1. Khái niệm về kiểu dữ liệu
2. Các kiểu dữ liệu trong C

II. Các kiểu dữ liệu cơ sở
1. Kiểu ký tự
2. Kiểu số nguyên
3. Kiểu số thực (số dấu phẩy động)

I.1. Khái niệm về kiểu dữ liệu

- Một kiểu dữ liệu là một tập giá trị mà có thể lưu trữ trong máy, trên đó xác định một số phép toán.
- Các kiểu dữ liệu trong C gồm có
  - Các kiểu dữ liệu cơ sở
    - Kiểu ký tự
    - Kiểu số nguyên
    - Kiểu số thực (số dấu phẩy động)

2. Các kiểu dữ liệu trong C

- Các kiểu dữ liệu có cấu trúc
  - Kiểu mảng
  - Kiểu xâu ký tự
  - Kiểu cấu trúc
  - Kiểu tệp
- Kiểu do người lập trình tự định nghĩa: Kiểu liệt kê

II. Các kiểu dữ liệu cơ sở (chuẩn)

1. Kiểu ký tự
2. Kiểu số nguyên
3. Kiểu số thực (kiểu số phẩy động)

II.1. Kiểu ký tự

- Kiểu ký tự được C định nghĩa với tên là char, gồm 256 ký tự trong bảng mã ASCII. Kiểu ký tự có kích thước 1 byte.
- Hằng ký tự là một ký tự cụ thể đặt giữa 2 dấu phẩy trên. Ví dụ: ’A’, ’b’, ’9’
- Một số hằng ký tự điều khiển:
  - ’\n’ New line, đặt con trỏ màn hình xuống đầu dòng tiếp theo
  - ’\t’ Tab
  - ’\b’ Backspace
  - ’\r’ Carriage return, đưa con trỏ màn hình về đầu dòng

II.1. Kiểu ký tự

- Hằng xâu ký tự là một dãy ký tự đặt giữa hai dấu nháy kép. Ví dụ: ”Nhap vao mot so”
- Kiểu ký tự có thể được dùng như kiểu số nguyên với các tên sau:
  - **char**: có giá trị -128 – 127
  - **unsigned char**: có giá trị 0 – 255
- Tất cả các ký tự đều lưu trữ trong bộ nhớ dưới dạng số là mã ASCII của ký tự đó.

II.2. Kiểu số nguyên

- Kiểu số nguyên được C++ định nghĩa với nhiều tên, được chia thành hai nhóm: kiểu số nguyên có dấu và kiểu số nguyên không dấu.
- Kiểu số nguyên có dấu gồm có:

Tên kiểu	Kích thước	Khoảng giá trị
short	2 byte	-32768 - 32767
int	2 hoặc 4 byte	-32768 - 32767
long	4 byte	$-2^{31} – 2^{31}-1$

2. Kiểu số nguyên

- Kiểu số nguyên không dấu gồm có:

Tên kiểu	Kích thước	Khoảng giá trị
unsigned short	2 byte	0 - 65535
unsigned int hoặc unsigned	2 hoặc 4 byte	0 - 65535
unsigned long	4 byte	$0 - 2^{32}-1$

- Các hằng số nguyên viết bình thường

Ví dụ: -45 2056 345

Chú ý: Các hằng số nguyên vượt ra ngoài khoảng của int được xem là hằng long (với trình biên dịch C++)

3. Kiểu số thực

Kiểu số thực được C định nghĩa với nhiều tên khác nhau:

Tên kiểu	Kích thước	Khoảng gía trị	Độ chính xác
float	4 byte	3.4E-38–3.4E38	7-8 chữ số
double	8 byte	1.7E-308–1.7E308	15-16 chữ số
long double	10 byte	3.4E-4932–1.1E4932	18-19 chữ số

Khoảng giá trị của mỗi kiểu số thực trên là giá trị tuyệt đối của số thực mà có thể lưu trữ trên máy. Giá trị nào có giá trị tuyệt đối nhỏ hơn cận dưới được xem như bằng 0.

3. Kiểu số thực

- Hằng số thực có 2 cách viết:
  - Dạng thập phân: gồm có phần nguyên, dấu chấm thập phân và phần thập phân.

  Ví dụ: 34.75 -124.25
  - Dạng mũ (dạng khoa học): gồm phần trị và phần mũ của cơ số 10, phần trị có thể là một số nguyên hoặc thực, phần mũ là một số nguyên âm hoặc dương. Hai phần cách nhau bởi chữ e hoặc E.

  Ví dụ: 125.34E-3 là số $125.34 \times 10^{-3} = 0.12534$

  0.12E3 là số $0.12 \times 10^{3} = 120$

  1E3 là số $10^{3} = 1000$

Chương 3. Các khai báo, biểu thức, khối lệnh

I. Các khai báo

II. Biểu thức

III. Khối lệnh

I.1. Khai báo sử dụng thư viện hàm

- Các trình biên dịch C có sẵn rất nhiều chương trình con (gọi là hàm), các hàm này để trong các thư viện hàm khác nhau. Muốn sử dụng hàm nào ta phải khai báo sử dụng thư viện hàm chứa hàm đó.
- Cú pháp khai báo như sau:

  #include<tên tệp header>

  hoặc #include "tên tệp header"

Tên tệp header của thư viện hàm có đuôi .h

Ví dụ: #include<stdio.h> //Khai báo sử dụng các chương trình vào/ra

I.2. Khai báo hằng

- Khai báo hằng là đặt tên cho một giá trị cụ thể
- Cú pháp khai báo hằng:

  #define Tên_hằng Giá_trị_của_hằng

Ví dụ: #define PI 3.141593

- Khai báo hằng có thể đặt bất kỳ đâu trong chương trình. Khi biên dịch chương trình, tất cả tên hằng được sử dụng sau dòng khai báo hằng sẽ được thay bằng giá trị của tên hằng.

I.3. Khai báo biến

- Biến là ô nhớ trong bộ nhớ trong (RAM) của máy tính dùng để cất chứa dữ liệu.
- Khai báo biến là đặt tên cho ô nhớ và xác định kiểu dữ liệu cho ô nhớ. Ô nhớ có kiểu dữ liệu nào thì chỉ chứa được giá trị của kiểu dữ liệu đó. Khai báo biến có thể để bất kỳ đâu trong chương trình.
- Cú pháp: **Tên_kiểu_dl Tên_biến;**

  Ví dụ: int a; //biến tên là a, có kiểu số nguyên int

  - Nếu có nhiều biến cùng kiểu thì có thể khai báo cùng nhau, giữa các tên biến phân tách nhau bởi dấu phẩy.

  Ví dụ: float a,b,c;

I.3. Khai báo biến (tiếp)

- Khi khai báo biến có thể khởi tạo giá trị ban đầu cho biến bằng đặt dấu bằng và một giá trị nào đó cách ngay sau tên biến.

  Ví dụ: int a,b=20,c,d=35;

II. Biểu thức

1. Biểu thức
2. Phép toán số học
3. Phép toán quan hệ và logic
4. Phép toán tăng giảm
5. Thứ tự ưu tiên của các phép toán
6. Các hàm số học
7. Câu lệnh gán và biểu thức gán
8. Biểu thức điều kiện
9. Chuyển đổi kiểu giá trị

II.1. Biểu thức

- Biểu thức là sự kết hợp các giá trị bằng các phép toán để có được một giá trị mới. Các giá trị đem ra kết hợp được gọi là toán hạng. Toán hạng có thể là hằng, biến, hàm.
- Biểu thức dùng để bảo máy tính thực hiện một tính toán nào đó để có được một giá trị mới.
- Mỗi biểu thức sẽ có một giá trị và nói chung cái gì có giá trị đều được coi là biểu thức.

II.1. Biểu thức (tiếp)

- Có hai loại biểu thức:
  - Biểu thức số: có giá trị là nguyên hoặc thực
  - Biểu thức logic: có giá trị là đúng (giá trị khác 0) hoặc sai (giá trị bằng 0)
- *Ví dụ:*

  (a+b+c)/2 (-b-sqrt(delta))/(2*a)

  (a+b) > 2*c

II.2. Phép toán số học

- Phép toán hai ngôi: + - * / %
  - % là phép lấy phần dư, ví dụ: 11%2 = 1
  - Phép chia hai số nguyên chỉ giữ lại phần nguyên

    Ví dụ: 11/2 = 5
- Phép toán một ngôi: dấu âm –

  Ví dụ -(a+b)
- Các phép toán số học tác động trên tất cả các kiểu dữ liệu cơ bản.

II.3. Phép toán so sánh và logic

- Các phép toán so sánh và logic cho ta giá trị đúng (có giá trị khác 0) hoặc sai (có giá trị bằng 0).
- Các phép toán so sánh gồm có:

Phép toán	Ý nghĩa
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
= =	Bằng (hai dấu bằng sát nhau)
!=	Khác nhau

II.3. Phép toán quan hệ và logic (tiếp)

- Các phép toán logic gồm có:

Phép toán	Ý nghĩa
!	Phủ định (NOT)
&&	Và (AND)
||	Hoặc (OR)

II.4. Phép toán tăng giảm

- C++ có hai phép toán một ngôi để tăng và giảm giá trị **của các biến** (có kiểu nguyên hoặc thực). Toán tử tăng ++ cộng 1 vào toán hạng của nó, toán tử giảm -- trừ toán hạng của nó đi 1.

  Ví dụ: giả sử biến n đang có giá trị là 8, sau phép tính ++n làm cho n có giá trị là 9, sau phép tính --n làm cho n có giá trị là 7.
- Phép toán ++ và -- có thể đứng trước hoặc sau toán hạng. Nếu đứng trước thì toán hạng của nó sẽ được tăng/giảm trước khi nó được sử dụng, nếu đứng sau thì toán hạng của nó sẽ được tăng/giảm sau khi nó được sử dụng.

II.5. Thứ tự ưu tiên của các phép toán

- Khi trong một biểu thức có chứa nhiều phép toán thì các phép toán được thực hiện theo thứ tự ưu tiên: Các phép toán có mức ưu tiên cao thực hiện trước, các phép toán cùng mức ưu tiên được thực hiện từ trái qua phải hoặc từ phải qua trái.
- Bảng thứ tự ưu tiên các phép toán: Các phép toán cùng loại cùng mức ưu tiên. Các phép toán loại 1 có mức ưu tiên cao nhất, rồi đến các phép toán loại 2, 3,… Các phép toán loại 2 (phép toán một ngôi), 14 (phép toán điều kiện) và 15 (phép toán gán) kết hợp từ phải qua trái, các phép toán còn lại kết hợp từ trái qua phải.

II.5. Thứ tự ưu tiên của các phép toán (tiếp)

TT	Loại phép toán	Phép toán	Ý nghĩa
1	Cao nhất	( ) [ ] -> . ::	Lời gọi hàm, dấu ngoặc Truy nhập phần tử mảng Truy nhập gián tiếp Truy nhập trực tiếp Truy nhập tên miền
2	Phép toán 1 ngôi	**!** ~ + - ++ --	Phủ định (NOT) Đảo bit Dấu dương Dấu âm Toán tử tăng Toán tử giảm

II.5. Thứ tự ưu tiên của các phép toán (*tiếp*)

TT	Loại phép toán	Phép toán	Ý nghĩa
2	Phép toán 1 ngôi	& * sizeof new delete (Kiểu dl)	Lấy địa chỉ biến Truy nhập qua con trỏ Cho kích thước toán hạng Cấp phát bộ nhớ động Giải phóng bộ nhớ Phép ép kiểu dữ liệu
3	Phép toán truy nhập thành viên	**.*** **->***	
4	Phép toán nhân	***** **/** **%**	Nhân Chia Chia lấy phần dư

5. Thứ tự ưu tiên của các phép toán (*tiếp*)

TT	Loại phép toán	Phép toán	Ý nghĩa
5	Phép toán cộng	+ -	Cộng Trừ
6	Phép toán dịch bit	>> <<	Dịch phải Dịch trái
7	Phép toán quan hệ	< <= > >=	Nhỏ hơn Nhỏ hơn hoặc bằng Lớn hơn Lớn hơn hoặc bằng
8	Phép toán so sánh bằng	== **!=**	Bằng Khác nhau

5. Thứ tự ưu tiên của các phép toán (*tiếp*)

TT	Loại phép toán	Phép toán	Ý nghĩa
9	Phép toán về bit	&	Phép AND bit
10	Phép toán về bit	^	Phép XOR bit
11	Phép toán về bit	|	Phép OR bit
12	Phép toán logic	**&&**	Phép AND logic
13	Phép toán logic	||	Phép OR logic
14	Phép toán điều kiện	**? :**	Ví dụ: a ? x : y //nếu a đúng thì bằng x, còn không bằng y

5. Thứ tự ưu tiên của các phép toán (*tiếp*)

TT	Loại phép toán	Phép toán	Ý nghĩa
15	Phép toán gán	=	Phép gán đơn giản
		*=	Phép gán nhân
		/=	Phép gán chia
		%=	Phép gán chia lấy phần dư
		+=	Phép gán cộng
		-=	Phép gán trừ
		&=	Phép gán AND bit
		^=	Phép gán XOR bit
		|=	Phép gán OR bit
		<<=	Phép gán dịch trái bit
		>>=	Phép gán dịch phải bit
16	Dấu phẩy	,	

II.6. Một số hàm toán học cơ bản

Các hàm số học nằm trong thư viện hàm math, muốn sử dụng các hàm này ta phải khai báo: #include<math.h>

Dưới đây là một số hàm số học hay dùng:

Tên hàm	Ý nghĩa
cos(x)	Cho cos(x)
sin(x)	Cho sin(x)
acos(x)	Cho arccos(x)
asin(x)	Cho arcsin(x)

6. Các hàm toán học cơ bản (tiếp)

Tên hàm	Ý nghĩa
tan(x)	Cho tgx
fabs(x)	Cho |x|
exp(x)	e^x
log(x)	Cho lnx
log10(x)	Cho $\log_{10}x$
pow(y,x)	Cho y^x
sqrt(x)	Cho căn bậc 2 của x

II.7. Câu lệnh gán và biểu thức gán

- Câu lệnh gán
  - Để đưa giá trị vào các biến tại thời điểm lập trình ta sử dụng lệnh gán. Có lệnh gán đơn giản và lệnh gán phức hợp.
  - Lệnh gán đơn giản có dạng: Biến = Biểu thức;

    Lệnh gán này đưa giá trị của biểu thức bên phải vào biến bên trái. Vế trái của phép gán chỉ có thể là biến và chỉ một mà thôi.

    Ví dụ: a = 2*x*x + 3*x + 1;

II.7. Câu lệnh gán và biểu thức gán (tiếp)

- Câu lệnh gán
  - Lệnh gán phức hợp có dạng:

    Biến Phép_toán= Biểu thức;

    Phép toán để ngay trước dấu bằng, có thể là các phép toán số học hoặc các phép toán về bit.

    Ví dụ: a += 2;

    Lệnh gán này đem giá trị của biến kết hợp với giá trị của biểu thức theo phép toán rồi đưa kết quả vào biến, tức là thực hiện phép toán trước rồi mới gán.

    a *= 5; //lệnh này tương đương với lệnh a = a*5;

II.7. Câu lệnh gán và biểu thức gán (tiếp)

- Biểu thức gán
  - Biểu thức gán là biểu thức có dạng:

    v = e

    (*Sau biểu thức gán không có dấu chấm phẩy*)

    trong đó v là một biến, e là một biểu thức.
  - Biểu thức gán thực hiện gán e vào v. Giá trị của biểu thức gán là giá trị của biểu thức e, kiểu của biểu thức gán là kiểu của biến v. Biểu thức gán được sử dụng như bất kỳ biểu thức khác, chẳng hạn đem gán giá trị của nó vào biến.

    Ví dụ: sau lệnh a = b = 5; thì a và b sẽ bằng 5 vì biểu thức gán đưa 5 vào b còn lệnh gán đưa giá trị của biểu thức gán b=5 vào a.

II.8. Biểu thức điều kiện

- Biểu thức điều kiện là biểu thức có dạng:

  e1 ? e2 : e3

  trong đó e1, e2, e3 là các biểu thức nào đó.
- Giá trị của biểu thức điều kiện bằng giá trị của e2 nếu e1 đúng (có giá trị khác 0) và bằng giá trị của e3 nếu e1 sai (có giá trị bằng 0).
- Biểu thức điều kiện thực sự là một biểu thức, bởi vậy ta có thể sử dụng nó như bất kỳ một biểu thức nào khác.

  Ví dụ: biểu thức (a > b) ? a : b sẽ cho giá trị a nếu a lớn hơn b, còn không cho giá trị b.

II.9. Chuyển đổi kiểu giá trị

- Việc chuyển đổi kiểu giá trị thường diễn một cách tự động trong hai trường hợp sau:
  - Khi biểu thức có các toán hạng khác kiểu
  - Khi gán một giá trị kiểu này cho một biến kiểu khác.
- 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. Kết quả thu được một giá trị có kiểu cao hơn.

  Ví dụ: giữa int và long thì int chuyển thành long

  giữa int và float thì int chuyển thành float

9. Chuyển đổi kiểu giá trị (tiếp)

- Chuyển đổi kiểu khi gán: Giá trị của vế phải được chuyển sang kiểu của vế trái.
- Ta cũng có thể thực hiện chuyển đổi kiểu theo ý muốn bằng toán tử ép kiểu, có dạng: (Tên kiểu muốn ép) Biểu_thức

  Ví dụ: (int) x (float)(a+b)

III. Khối lệnh

- Nhiều lệnh đặt giữa dấu ngoặc { và } tạo thành một khối lệnh.

```
{
   a=2;
   b=3;
   cout<<a<<' '<<b;
}
```

- C++ coi một khối lệnh như một câu lệnh riêng lẻ. Bởi vậy chỗ nào viết được một câu lệnh thì chỗ đó viết cũng đặt được một khối lệnh. Sau dấu ngoặc } của khối lệnh không có dấu chấm phẩy.

III. Khối lệnh (tiếp)

- Bên trong một khối lệnh có thể chứa các khối lệnh khác. Sự lồng nhau này không bị hạn chế. Lưu ý rằng thân của một hàm cũng là một khối lệnh, đó là khối lệnh chứa các khối lệnh bên trong nó và không khối lệnh nào chứa nó.
- Các biến không chỉ khai báo ở đầu một hàm mà có thể khai báo ở đầu một khối lệnh. Biến được khai báo trong một khối lệnh thì chỉ có phạm vi hoạt động trong khối lệnh đó. Khi máy bắt đầu thực hiện khối lệnh thì các biến khai báo bên trong nó mới 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 sẽ lập tức biến mất ngay sau khi máy ra khỏi khối lệnh.

Chương 4. Lệnh vào/ra dữ liệu và các cấu trúc điều khiển chương trình

I. Lệnh vào/ra dữ liệu

II. Lệnh lựa chọn

III. Lệnh lặp

IV. Lệnh break và continue

I. Lệnh vào/ra dữ liệu

1. Khai báo thư viện hàm vào/ra dữ liệu
2. Lệnh lấy dữ liệu vào từ bàn phím
3. Lệnh đưa dữ liệu ra màn hình
4. Kết hợp giữa lệnh printf và scanf để tổ chức lấy dữ liệu vào từ bàn phím

I.1. Khai báo thư viện hàm vào/ra dữ liệu

- Để có thể sử dụng các lệnh vào/ra dữ liệu của C khi lập trình trên DOS ta phải khai báo sử dụng thư viện hàm stdio:

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


I.2. Lệnh lấy dữ liệu vào từ bàn phím

- Để lấy dữ liệu từ bàn phím vào biến ta dùng lệnh scanf theo cú pháp sau:

```
scanf(đặc tả kiểu dl, địa chỉ các ô nhớ);
```

Trong đó: 1) đặc tả kiểu dl là hằng xâu ký tự điều khiển chỉ chứa các đặc tả chuyển dạng dữ liệu, mỗi đặc tả tương ứng với một địa chỉ ô nhớ;

2) địa chỉ các ô nhớ phân tách nhau bởi dấu chấm phẩy. Sử dụng toán tử & để lấy địa chỉ ô nhớ của biến, ví dụ &a

I.2. Lệnh lấy dữ liệu vào từ bàn phím

3) Đặc tả chuyển dạng dữ liệu có cấu trúc chung như sau:

%[*][w]Ký tự chuyển dạng

- Nếu có dấu * thì trường vào vẫn được dò đọc bình thường nhưng giá trị của nó không được lưu vào bộ nhớ. Đặc tả chứa dấu * sẽ không có ô nhớ tương ứng.

- w là một số xác định chiều dài cực đại của trường vào.

Nếu không có tham số w hoặc nếu tham số này lớn hơn hoặc bằng độ dài trường vào thì toàn bộ trường vào sẽ được đọc, nội dung của nó được dịch và được đưa vào ô nhớ tương ứng.

Nếu w nhỏ hơn độ dài của trường vào tương ứng thì chỉ phần đầu của trường vào có độ dài bằng w được đọc, được dịch và được gán vào ô nhớ tương ứng. Phần còn lại sẽ được dùng cho đặc tả tiếp theo. Ví dụ: vdch4_01.cpp

I.2. Lệnh lấy dữ liệu vào từ bàn phím

4) Ký tự chuyển dạng xác định cách thức dò đọc dữ liệu trên dòng vào cũng như phương pháp chuyển dịch thông tin đọc được trước khi gán nó cho các địa chỉ tương ứng.

Các ký tự chuyển dạng dữ liệu dùng cho scanf

Ký tự chuyển dạng	Ý nghĩa
c	Đọc một ký tự, đối tương ứng là ô nhớ kiểu char
d	Đọc một giá trị int, đối tương ứng là ô nhớ kiểu int
ld	Đọc một giá trị long, đối tương ứng là ô nhớ kiểu long
o	Đọc một giá trị kiểu int hệ 8, đối tương ứng là ô nhớ kiểu int
lo	Đọc một giá trị kiểu long hệ 8, đối tương ứng là ô nhớ kiểu long
x	Đọc một giá trị kiểu int hệ 16, đối tương ứng là ô nhớ kiểu int
lx	Đọc một giá trị kiểu long hệ 16, đối tương ứng là ô nhớ kiểu long
f hoặc e	Đọc một giá trị kiểu float, đối tương ứng là ô nhớ kiểu float
lf hoặc le	Đọc một giá trị kiểu double, đối tương ứng là ô nhớ kiểu double
s	Đọc một xâu ký tự, đối tương ứng là mảng các ô nhớ kiểu char

Các ký tự chuyển dạng dữ liệu dùng cho scanf

Ký tự chuyển dạng	Ý nghĩa
[dãy ký tự]	Đọc các ký tự cho tới khi gặp một ký tự không thuộc tập các ký tự trong hai dấu [ ]. Đối tương ứng là địa chỉ của mảng các ô nhớ kiểu char. Khoảng trắng cũng được xem là ký tự.
[^dãy ký tự]	Đọc các ký tự cho tới khi gặp một ký tự thuộc tập các ký tự trong hai dấu [ ]. Đối tương ứng là địa chỉ của mảng các ô nhớ kiểu char. Khoảng trắng cũng được xem là ký tự.

I.3. Lệnh đưa dữ liệu ra màn hình

- Cú pháp:

  printf(dk,các dữ liệu cần đưa ra);

  Trong đó: 1) dk là hằng xâu ký tự điều khiển có chứa:

  + Các ký tự điều khiển, ví dụ như '\n', '\t', '\b'

  + Các đặc tả chuyển dạng và tạo khuôn dữ liệu, mỗi đặc tả dùng cho một dữ liệu tương ứng cần đưa ra màn hình.

  + Các ký tự thông thường.

2) Các dữ liệu cần đưa ra có thể là hằng, biến, biểu thức. Có bao nhiêu dữ liệu đưa ra thì phải có bấy nhiêu đặc tả chuyển dạng.

Đặc tả chuyển dạng dữ liệu

- Cấu trúc chung:

  %[-][fw][.pp]Ký tự chuyển dạng

  - Nếu không có dấu trừ - thì dữ liệu được căn phải trong số chỗ trên màn hình dành cho dữ liệu, còn thừa chỗ để trống. Với dữ liệu là số, nếu fw bắt đầu bằng số 0 thì các chỗ trống sẽ được điền đầy bằng các số 0.

  - Nếu có dấu trừ thì dữ liệu sẽ được căn trái, các chỗ thừa luôn để trống.

  Ví dụ trên máy với dữ liệu cần đưa ra là -2503

Đặc tả chuyển dạng dữ liệu (tiếp)

- Cấu trúc chung:

  %[-][fw][.pp]Ký tự chuyển dạng

  - fw là số nguyên xác định số chỗ trên màn hình dành cho dữ liệu đưa ra. Nếu không có fw hoặc nếu fw nhỏ hơn độ dài thực tế của dữ liệu thì số chỗ trên màn hình dành cho dữ liệu sẽ bằng độ dài của dữ liệu.

  - pp là số nguyên xác định số chữ số sau dấu chấm thập phân. pp chỉ dùng cho dữ liệu là số thực.

Các ký tự chuyển dạng dữ liệu dùng cho printf

Ký tự chuyển dạng	Kiểu dữ liệu	Các chuyển dạng
c	char	Dữ liệu được coi là ký tự
d hoặc i	int	Dữ liệu được coi là số nguyên có dấu
ld hoặc li	long	Dữ liệu được coi là số nguyên có dấu
u	int	Dữ liệu được coi là số nguyên không dấu
o	int	Dữ liệu được coi là số hệ 8 không dấu
lo	long	Dữ liệu được coi là số hệ 8 không dấu
x	int	Dữ liệu được coi là số hệ 16 không dấu
lx	long	Dữ liệu được coi là số hệ 16 không dấu
f	float/double	Dữ liệu được coi là số thực dạng thập phân
e	float/double	Dữ liệu được coi là số thực dạng mũ
s	Xâu ký tự	Dữ liệu được coi là xâu ký tự

I.4. Kết hợp giữa lệnh printf và scanf để tổ chức lấy dữ liệu vào từ bàn phím

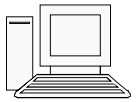
- Trước mỗi lệnh nhập dữ liệu scanf ta nên dùng lệnh printf để đưa ra một lời nhắc nhập vào dữ liệu gì.

  printf(“Lời nhắc: ”); scanf( );

Một chương trình C đơn giản

Ví dụ 4.1:

Chương trình này lấy vào bán kính của một hình tròn, sau đó tính và đưa ra màn diện tích và chu vi của hình tròn.



II. Lệnh lựa chọn

II.1. Lệnh kiểm tra điều kiện if

- Lệnh kiểm tra điều kiện là để bảo máy kiểm tra một điều kiện, nếu đúng thì làm công việc này, nếu sai thì làm công việc khác. Biểu thức điều kiện là một biểu thức logic có giá trị đúng (khác 0) hoặc sai (bằng 0).
- Lệnh này có 2 dạng:

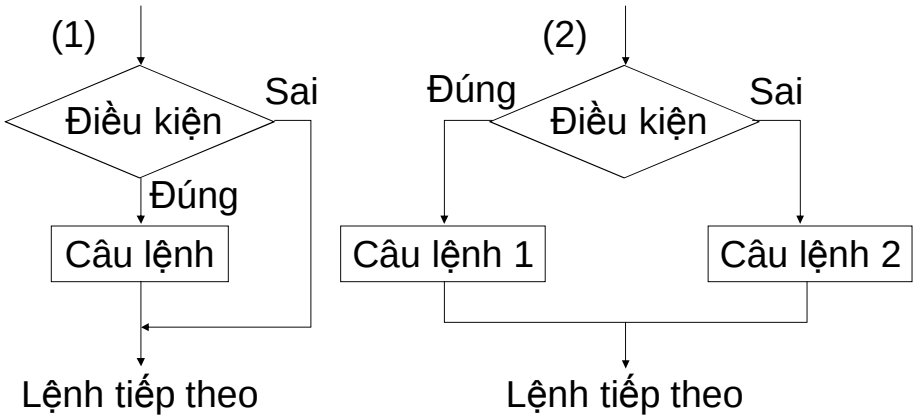
  (1) if (điều kiện) Câu lệnh;

  (2) if (điều kiện) Câu_lệnh_1; else Câu_lệnh_2;

  trong đó Câu_lệnh có thể là một câu lệnh đơn lẻ hoặc một khối lệnh. Lưu ý là Điều kiện phải đặt trong ngoặc và sau Câu_lệnh_1 vẫn phải có dấu chấm phẩy.

II.1. Lệnh kiểm tra điều kiện if (tiếp)

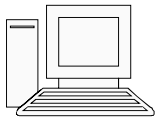
- Lưu đồ thực hiện lệnh dạng (1) và (2) như sau:



II.1. Lệnh kiểm tra điều kiện if (tiếp)

- Ví dụ 4.1:
  Viết chương trình nhập vào một số thực, kiểm tra nếu số đó dương thì đưa ra màn hình căn bậc 2 của số đó, nếu âm thì đưa ra thông báo "Số âm không có căn bậc 2".

```
//Khai bao su dung thu vien chuong trinh
#include<stdio.h>
#include<math.h>
int main()
{
   float a;
   printf("Nhap vao mot so: "); scanf("%f",&a);
   if (a>=0) printf("Can bac 2 bang: %6.2f",sqrt(a));
   else printf("So am khong co can bac 2");
   return 0;
}
```



II.2. Lệnh thử và rẽ nhánh switch

- Khi cần kiểm tra giá trị của một biểu thức xem có bằng một giá trị nào trong nhiều giá trị không ta dùng lệnh switch.
- Cú pháp: có 2 dạng

(1)

```
switch (Biểu thức)        <- Không có chấm phẩy
 {
      case hằng1:
            Các câu lệnh;   <- Các lệnh ứng với hằng 1
            break;          <- Để thoát khỏi switch
      case hằng2:
            Các câu lệnh;   <- Các lệnh ứng với hằng 2
            break;
      ……
      case hằngN:
            Các câu lệnh;   <- Các lệnh ứng với hằng N
            break;
 }                          <- Không có chấm phẩy
```


II.2. Lệnh thử và rẽ nhánh switch (tiếp)

(2)

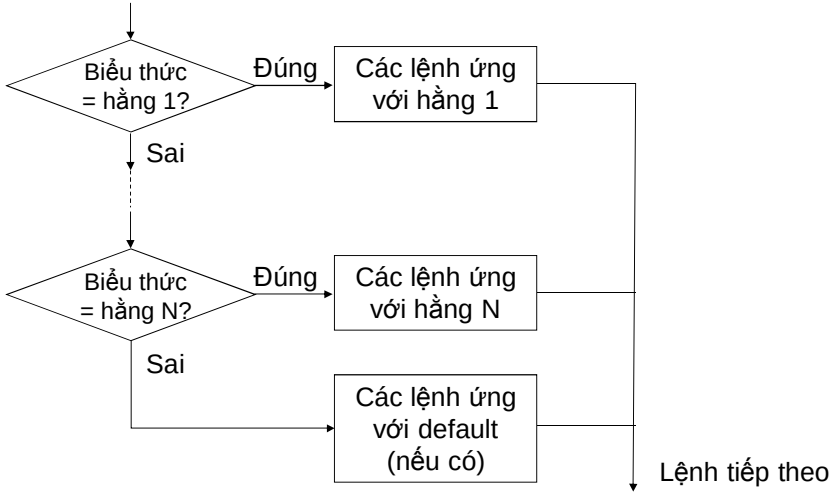
```
switch (Biểu thức)        <- Không có dấu chấm phẩy
 {
      case hằng1:
        Các câu lệnh;
        break;              <- Các lệnh ứng với hằng 1
      case hằng2:           <- Để thoát khỏi switch
        Các câu lệnh;
        break;              <- Các lệnh ứng với hằng 2
        ……
      case hằngN:
        Các câu lệnh;
        break;              <- Các lệnh ứng với hằng N
      default:
        Các câu lệnh;
        break;              <- Các lệnh ứng với default
 }                          <- Không có dấu chấm phẩy
```

II.2. Lệnh thử và rẽ nhánh switch (tiếp)

- Biểu thức sau từ khoá switch phải đặt trong ngoặc đơn.
- Biểu thức và các hằng phải cùng kiểu và phải là kiểu số nguyên hoặc ký tự.
- Các hằng có thể là một giá trị hằng hoặc biểu thức hằng (các hằng kết hợp với nhau). Sau các hằng phải có dấu hai chấm.
- Trước mỗi hằng phải có từ khoá case, tức là không thể có nhiều hằng chung một từ khoá case.
- Nếu muốn nhiều hằng cùng chung một câu lệnh thì các hằng này để gần nhau và chỉ viết các lệnh cùng câu lệnh break ở hằng dưới cùng.

II.2. Lệnh thử và rẽ nhánh switch (tiếp)

Lưu đồ thực hiện lệnh switch như sau:



II.2. Lệnh thử và rẽ nhánh switch (tiếp)

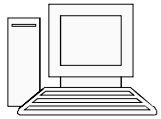
Ví dụ 4.2:

Viết chương trình nhập vào tháng và năm, cho biết tháng trong năm đó có bao nhiêu ngày?

(*Chương trình trang sau)*

II.2. Lệnh thử và rẽ nhánh switch (tiếp)

```
//Khai bao su dung thu vien chuong trinh
#include<stdio.h>
int main()
{
    int thang,nam;
    printf("Nhap vao thang: ");scanf("%d",&thang);
    printf("Nhap vao nam: ");scanf("%d",&nam);
    switch(thang)
    {
          case 1:
          case 3:
          case 5:
          case 7:
          case 8:
          case 10:
          case 12:
                    printf("Thang nay co 31 ngay!");
                    break;
          case 4:
          case 6:
          case 9:
          case 11:
                    printf("Thang nay co 30 ngay!");
                    break;
          case 2:
                    if(nam%4==0) printf("Thang nay co 29 ngay!");
                    else printf("Thang nay co 28 ngay!");
                    break;
          default:
                    printf("Thang nhap vao ko dung!");
                    break;
    }
    return 0;
}
```



III. Lệnh lặp

1. Lệnh lặp với số lần lặp xác định for
2. Lệnh lặp với lần lặp không xác định

III.1. Lệnh lặp với số lần xác định for

- Cú pháp:

for (Biểu thức khởi tạo;Biểu thức kiểm tra; Biểu thức tăng/giảm)
    Câu lệnh hoặc Khối lệnh

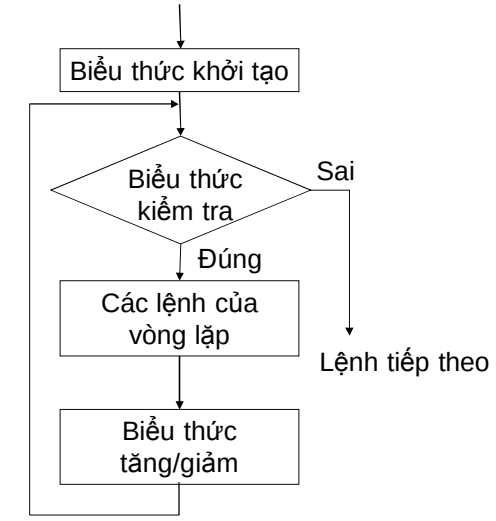
  - Biểu thức khởi tạo dùng để khởi tạo giá trị ban đầu cho biến điều khiển vòng lặp và chỉ được thực hiện duy nhất một lần khi bắt đầu vào vòng lặp for.

III.1. Lệnh lặp với số lần xác định for (tiếp)

  - Biểu thức kiểm tra dùng để kiểm tra giá trị của biến điều khiển xem còn tiếp tục lặp hay kết thúc. Biểu thức kiểm tra thường là biểu thức logic có giá trị đúng hoặc sai, khi có giá trị đúng thì vẫn lặp, khi có giá trị sai thì kết thúc.
  - Biểu thức tăng/giảm dùng để thay đổi biến điều khiển theo chiều tăng hoặc giảm.

III.1. Lệnh lặp với số lần xác định for (tiếp)

- Lưu đồ thực hiện lệnh for như bên:
- Ba biểu thức trong lệnh for có thể không có nhưng hai dấu chấm phẩy không thể thiếu. Khi không viết biểu thức kiểm tra thì mặc định biểu thức kiểm tra có giá trị true, điều này làm cho vòng lặp lặp mãi.



III.1. Lệnh lặp với số lần xác định for (tiếp)

- *Ví dụ:*

```
for (i=1;i<=10;i++)
    printf("%d\n",i);
for (i=10;i<=20;i+=2)
 {
    printf("%d",i);
    printf("\n");
 }
```

Không có dấu chấm phẩy

III.1. Lệnh lặp với số lần xác định for (tiếp)

Ví dụ: 1) Tính S = 1 + 2 + 3 + … + N (tính theo phương pháp cộng dồn)

BTVN:

1) Viết chương trình tính gần đúng số π theo công thức sau (với n số hạng đầu tiên):

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots + \frac{(-1)^n}{2n+1}$$

2) Tính n!

III.2. Lệnh lặp với số lần lặp không xác định

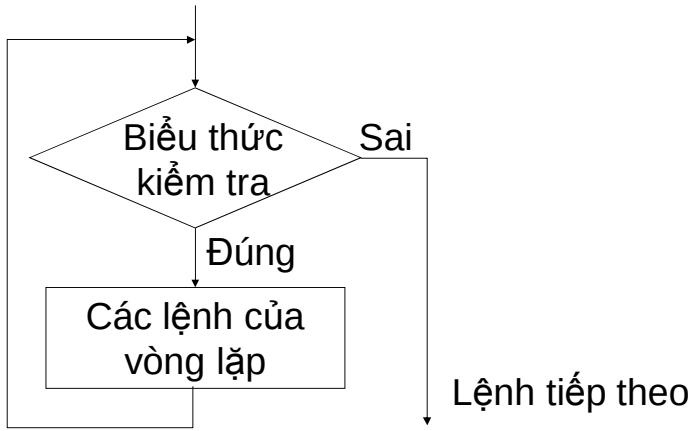
- Lệnh lặp kiểm tra điều kiện trước while

```
while (Biểu thức kiểm tra)
    Câu lệnh;
```

Không có dấu chấm phẩy

III.2. Lệnh lặp với số lần lặp không xác định (tiếp)

- Lưu đồ thực hiện lệnh while



III.2. Lệnh lặp với số lần lặp không xác định (tiếp)

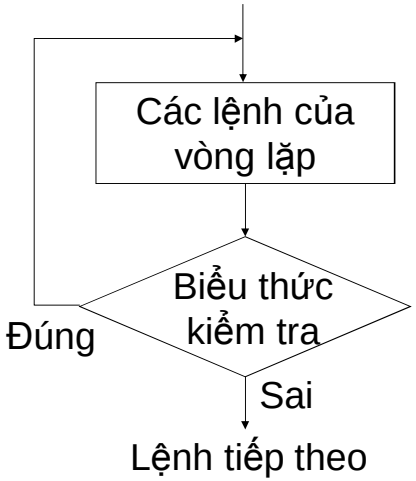
- Lệnh lặp kiểm tra điều kiện sau do-while

```
do
  Câu lệnh;
while (Biểu thức kiểm tra);
```

Không có dấu chấm phẩy

III.2. Lệnh lặp với số lần lặp không xác định (tiếp)

- Lưu đồ thực hiện lệnh do … while



III.2. Lệnh lặp với số lần lặp không xác định (tiếp)

Ví dụ: Tìm USCLN(a, b)

BTVN:

1) Viết chương trình tính e^x theo công thức:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots$$

Với độ chính xác 10^{-5}, tức là ta cần chọn n sao cho

$$\left|\frac{x^n}{n!}\right| < 0.00001$$

2) Làm lại bài tính gần đúng số PI với độ chính xác 10^{-4}.

IV. Lệnh break và continue

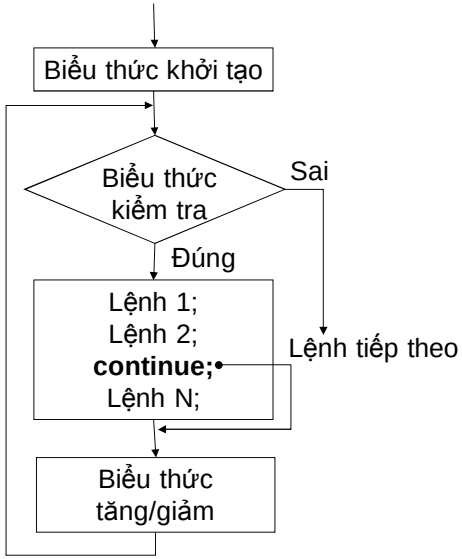
- Lệnh break được dùng để thoát khỏi lệnh for, while, do-while và switch. Nếu các lệnh này lồng nhau thì lệnh break thoát khỏi lệnh bên trong nhất chứa nó.
- Với lệnh break ta có thể thoát khỏi vòng lặp từ một điểm bất kỳ bên trong vòng lặp mà không dùng đến điều kiện kết thúc vòng lặp.
- *Ví dụ:* Viết chương trình nhập vào một số nguyên dương, cho biết số này có phải là số nguyên tố không?

IV. Lệnh break và continue

- Lệnh continue chỉ dùng với các lệnh lặp for, while và do-while.
- Lệnh continue không làm thoát khỏi lệnh lặp mà làm cho lệnh lặp bỏ qua các lệnh sau lệnh continue để thực hiện vòng lặp tiếp theo.

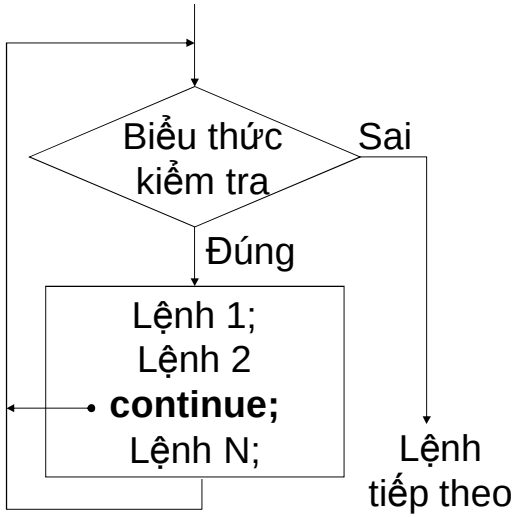
Lệnh continue (tiếp)

- Tác động của lệnh continue đối với lệnh for.



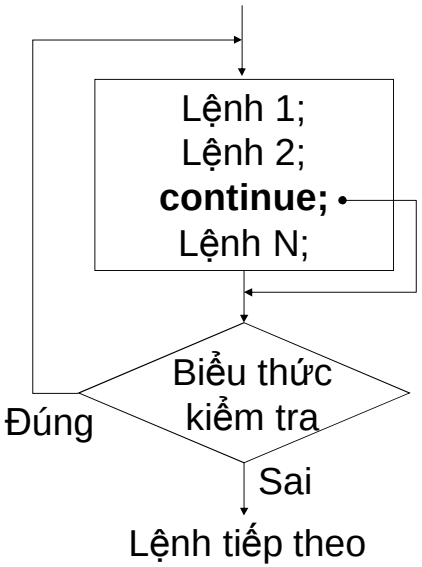
Lệnh continue (tiếp)

- Tác động của lệnh continue đối với lệnh while.



Lệnh continue (tiếp)

- Tác động của lệnh continue đối với lệnh do-while.



Chương 5. Kiểu mảng và xâu ký tự

I. Mảng

I.1. Khái niệm về kiểu mảng

- Mảng là một nhóm các biến nằm cạnh nhau có cùng kiểu, cùng tên. Mỗi biến được gọi là một phần tử. Các phần tử của mảng được truy nhập trực tiếp thông qua tên biến mảng và chỉ số.
- Số phần tử của mảng được xác định ngay từ khi định nghĩa ra mảng. Đây là điểm hạn chế của mảng bởi vì nếu không dùng hết các biến của mảng sẽ gây lãng phí bộ nhớ.

I.2. Khai báo biến mảng một chiều

- Khai báo biến mảng là xác định tên biến mảng, kiểu phần tử, số chiều và kích thước mỗi chiều.
- Cú pháp khai báo biến mảng một chiều:

  Kiểu_phần_tử Tên_biến_mảng[Kích thước];

  trong đó kích thước là số phần tử của mảng, phải cho dưới dạng hằng hoặc biểu thức hằng. Kiểu phần tử có thể là bất kỳ kiểu nào.

  Ví dụ: int a[5];

  Ví dụ này định nghĩa một biến mảng có tên là a, kiểu phần tử là int, số chiều là một và kích thước (số phần tử của mảng) là 5.

I.3. Truy nhập các phần tử của mảng một chiều

- Các phần tử của mảng được đánh số. Các số này gọi là chỉ số. Phần tử đầu tiên có chỉ số là 0, phần tử thứ 2 có chỉ số là 1,… Mảng có kích thước n thì phần tử cuối cùng có chỉ số n-1.
- *Ví dụ:* nếu ta định nghĩa một biến mảng

  int a[5];

  thì ta được một biến mảng tên là a có 5 phần tử, phần tử đầu tiên có chỉ số là 0, phần tử thứ 5 có chỉ số là 4.

I.3. Truy nhập các phần tử của mảng một chiều

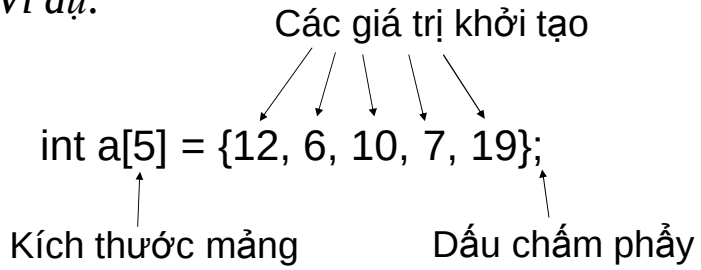
- Mỗi phần tử của mảng có thể truy nhập trực tiếp thông qua tên biến mảng và chỉ số của nó đặt trong ngoặc vuông []. Chỉ số của phần tử có thể cho dưới dạng hằng hoặc biểu thức.
- *Ví dụ:* 5 phần tử của mảng a ở ví dụ trên có tên là a[0], a[1],… Ta có thể dùng các lệnh sau:

  a[0]=100; cout<<a[1];

  for(int i=0;i<5;++i) scanf(“%d”,&a[i]);

I.4. Khởi tạo mảng một chiều

- Ta có thể khởi tạo giá trị cho các phần tử của mảng ngay khi khai báo bằng cách liệt kê các giá trị khởi tạo đặt trong ngoặc {}.
- *Ví dụ:*



I.4. Khởi tạo mảng một chiều (tiếp)

- Nếu số giá trị khởi tạo ít hơn kích thước mảng thì các phần tử còn lại sẽ được khởi tạo bằng 0. Nếu số giá trị khởi tạo lớn hơn kích thước mảng thì trình biên dịch sẽ báo lỗi.

  Ví dụ: int a[3] = {6,8}; //a[0]=6, a[1]=8, a[2]=0

  int a[2] = {8, 6, 9}; //Báo lỗi
- Với những mảng được khởi tạo có thể không cần xác định kích thước mảng. Khi đó trình biên dịch sẽ đếm số giá trị khởi tạo và dùng số đó làm kích thước mảng. *Ví dụ:*

  int a[] = {3, 5, 8}; //sẽ được mảng có kích thước là 3

I.5. Mảng nhiều chiều

- Mảng một chiều là mảng mà các phần tử của nó được truy nhập qua một chỉ số. Mảng nhiều chiều là mảng mà các phần tử được truy nhập qua nhiều chỉ số.
- C cho phép khai báo các mảng nhiều chiều với kích thước mỗi chiều có thể khác nhau. Cú pháp chung như sau:

Kiểu Tên_biến_mảng[Kích thước chiều 1][Kích thước chiều 2]…;

- Ví dụ:

```
int a[4][3];
```

Lưu ý là mỗi chiều phải được bao bởi cặp ngoặc []

I.5. Mảng nhiều chiều (tiếp)

- Để truy nhập phần tử của mảng m chiều thì ta phải dùng m chỉ số. Chỉ số của mỗi chiều có giá trị từ 0 đến kích thước của chiều đó trừ đi 1. Cú pháp chung như sau:

Tên_biến_mảng[chỉ số chiều 1][Chỉ số chiều 2]…

- Mảng 2 chiều có thể xem như là mảng một chiều có các phần tử là một mảng một chiều.
- Ta cũng có thể khởi tạo giá trị cho các phần tử của mảng nhiều chiều ngay khi định nghĩa. Ví dụ:

```
int a[2][3] = {{5, 7, 9},{3, 6, 7}};
```


I.6. Chú ý về chỉ số của phần tử mảng

- Trình biên dịch C sẽ không báo lỗi khi chỉ số dùng để truy nhập phần tử của mảng nằm ngoài khoảng cho phép, tức là nhỏ hơn 0 hoặc lớn hơn kích thước mảng trừ 1. Điều này rất nguy hiểm bởi vì nếu ta ghi dữ liệu vào phần tử mảng với chỉ số nằm ngoài khoảng cho phép thì có thể ghi đè lên dữ liệu của các chương trình khác đang chạy hoặc chính chương trình của ta.

I.7. Vào/ra với biến mảng

- Không dùng được lệnh printf và scanf với cả biến mảng, chỉ dùng được với từng phần tử của mảng. Ví dụ:

```
int a[5];
for(int i=0;i<5;++i)
    {printf("Nhap vao phan tu thu %d: ", i+1);
     scanf("%d",&a[i]);
    }
for(int i=0;i<5;++i) printf("%d  ",a[i]);
```

Bài tập

1. Cho dãy số nguyên a_1, a_2, a_3,…, a_n. Sắp xếp dãy số tăng dần từ trái qua phải.
2. Cho dãy số a_1, a_2, a_3,…, a_n. Tính tổng và trung bình cộng các số dương mà chia hết cho 3.
3. Cho ma trận nguyên có m hàng, n cột. Tính tổng và trung bình cộng các phần tử dương chẵn. Đưa ma trận đã nhập và các kết quả ra màn hình. Ma trận đưa ra phải theo hàng, cột.

II. Xâu ký tự

II.1. Khái niệm về kiểu xâu ký tự

- Xâu ký tự là một dãy ký tự có ký tự cuối cùng là ký tự rỗng. Ký tự rỗng có giá trị số là 0 và viết là '\0'.
- Xâu ký tự được C lưu trữ như một mảng ký tự, nó cho phép truy nhập vào từng ký tự của xâu như truy nhập vào từng phần tử của mảng. Tuy nhiên, trong một số trường hợp C xem xâu ký tự như những kiểu dữ liệu cơ bản. Ví dụ, có thể nhập vào và đưa ra cả biến xâu bằng lệnh scanf và printf.

II.2. Khai báo biến xâu ký tự

- Khai báo biến xâu ký tự là xác định tên biến xâu và số ký tự cực đại có thể chứa trong biến xâu.
- Cú pháp khai báo biến xâu ký tự giống cú pháp khai báo biến mảng một chiều:

```
char Tên_biến_xâu[Kích thước];
```

Ví dụ:

```
char s[16];
```

trong đó số ký tự cực đại cho dưới dạng hằng hoặc biểu thức hằng.

- Biến xâu có thể chứa các xâu ký tự có độ dài khác nhau.

II.3. Khởi tạo biến xâu

- Khi định nghĩa biến xâu ta có thể khởi tạo cho nó. Dưới đây là 2 cách khởi tạo:
  - Khởi tạo như biến mảng:

    char str[6] = {'D', 'H', 'N', 'N', 'I', '\0'};
  - Khởi tạo bằng hằng xâu:

    char str[6] = "DHNNI";

    Hằng xâu là một dãy ký tự đặt giữa 2 dấu phẩy kép. Khi viết hằng xâu ta không viết ký tự '\0', ký tự này sẽ được trình biên dịch thêm vào. Hằng xâu rỗng là hằng xâu không có ký tự nào "".

II.3. Khởi tạo biến xâu (tiếp)

- Lưu ý là khi khởi tạo cho biến xâu bằng hằng xâu thì số ký tự cực đại của biến xâu phải lớn hơn số ký tự của hằng xâu ít nhất là 1, bởi vì trình biên dịch sẽ đưa thêm vào biến xâu một ký tự rỗng. Ví dụ:

  char str[5] = "DHNNI"; //Sai

  char str[6] = "DHNNI"; //Đúng
- Cũng giống như biến mảng, khi khởi tạo cho biến xâu thì có thể không cần xác định số ký tự cực đại, khi đó trình biên dịch sẽ xác định số ký tự cực đại bằng số ký tự của hằng xâu cộng thêm 1. Ví dụ:

  char str[] = "DHNNI";

II.4. Vào/ra với biến xâu

- Có thể dùng lệnh printf và scanf với cả biến xâu. Ví dụ:

  char str[11];

  printf(“%s”,str); scanf(“%s”,str);
- scanf chỉ nhập vào được các xâu ký tự không có dấu cách.
- Sử dụng gets(Biến xâu) để nhập vào xâu ký tự có cả dấu cách. Ví dụ: gets(str);

II.5. Các hàm chuẩn xử lý xâu ký tự

- C có một thư viện hàm làm việc với xâu ký tự là string.lib. Muốn sử dụng các hàm này ta phải khai báo sử dụng:

  #include<string.h>
- Hàm lấy độ dài của xâu: strlen(s) cho độ dài của xâu s (không tính ký tự '\0'). Ví dụ: strlen(“08T1A”) => 5
- Hàm copy xâu: strcpy(s1, s2) copy xâu s2 vào biến xâu s1, s2 có thể là hằng xâu hoặc biến xâu.

II.5. Các hàm chuẩn xử lý xâu ký tự (tiếp)

- Hàm nối xâu: strcat(s1,s2) nối xâu s2 vào cuối biến xâu s1, s2 có thể là hằng xâu hoặc biến xâu, biến xâu s1 phải có số ký tự cực đại đủ chứa các ký tự s2 khi thêm vào.
- Hàm so sánh xâu: strcmp(s1,s2) so sánh hai xâu s1 và s2 theo mã ASCII, có phân biệt chữ hoa chữ thường. Hàm trả về một giá trị int:

  < 0 nếu s1 < s2

  ==0 nếu s1 == s2

  > 0 nếu s1 > s2

II.5. Các hàm chuẩn xử lý xâu ký tự (tiếp)

- Hàm đảo xâu: strrev(s) đảo ngược các ký tự trong xâu s, đầu về cuối, cuối về đầu.
- Hàm chuyển chữ thường thành chữ hoa: strupr(s) chuyển các chữ cái thường trong xâu s thành chữ hoa, các chữ khác không thay đổi.
- Hàm chuyển chữ hoa thành chữ thường: strlwr(s) chuyển các chữ cái hoa trong xâu s thành chữ thường, các chữ khác không thay đổi.

II.6. Mảng xâu ký tự

- Một mảng xâu ký tự rất hay được sử dụng, chẳng hạn như dùng để lưu trữ danh sách tên, danh sách mật khẩu, danh sách tên tệp,…
- Để tạo mảng các biến xâu rỗng ta tạo một mảng hai chiều bởi vì xâu ký tự cũng là một mảng và mảng xâu ký tự thực chất là mảng của các mảng.
- Ví dụ: để lưu trữ 5 họ tên, mỗi họ tên có tối đa 20 ký tự ta định nghĩa mảng xâu như sau:

  char names[5][21]; Đoạn chương trình dưới đây cho phép người sử dụng nhập vào các họ tên để lưu trong mảng trên.

II.6. Mảng xâu ký tự (tiếp)

```
for(int i=0;i<5;++i)
  {
   printf("Nhap vao mot ho ten (an enter de thoat): “);
   gets(names[i]);
   if(strlen(names[i])==0)
   break;
  }
```

II.6. Mảng xâu ký tự (tiếp)

- Ta cũng có thể khởi tạo mảng xâu ngay khi định nghĩa giống như các mảng khác. Ví dụ:

```
char Thu[7][] =
{"Thu Hai", "Thu Ba",  "Thu Tu", "Thu Nam",
 "Thu Sau", "Thu Bay", "Chu Nhat"};
```


Bài tập chương 5

- Bài 1. Viết chương trình nhập vào một dãy n số nguyên, hãy sắp xếp dãy số này theo thứ tự không giảm bằng phương pháp sắp xếp chọn.
- Bài 2. Hình vuông kỳ ảo bậc n được định nghĩa là một ma trận vuông cấp n sao cho:
  - Chứa đủ n^2 số tự nhiên đầu tiên (1, 2, 3,…, n^2)
  - Tổng các số trên từng hàng bằng tổng các số trên từng cột bằng tổng các số trên đường chéo chính bằng tổng các số trên đường chéo phụ.

  Viết chương trình nhập vào số tự nhiên lẻ n, đưa ra màn hình một hình vuông kỳ ảo bậc n lẻ đó.

Bài tập chương 6 (tiếp)

Ví dụ dưới đây là 2 hình vuông kỳ ảo bậc 3 và bậc 5:

8	1	6
3	5	7
4	9	2

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Bài tập chương (tiếp)

- Bài 3. Viết chương trình nhập vào một số tự nhiên n, đưa ra màn hình số nhị phân tương ứng dưới dạng xâu ký tự.
- Bài 4. Hai từ x và y gọi là anagram với nhau nếu mỗi ký tự của từ này cũng có mặt trong từ kia (không phân biệt chữ hoa chữ thường) và hơn nữa số lượng từng loại ký tự xuất hiện trong hai từ là bằng nhau. Ví dụ các từ sau là anagram của nhau: read, dear, dare. Viết chương trình nhập vào 2 từ x và y rồi kiểm tra xem chúng có phải là anagram của nhau không.

Chương 6. Kiểu cấu trúc và kiểu liệt kê

I. Kiểu cấu trúc (struct)

II. Kiểu liệt kê (enum)

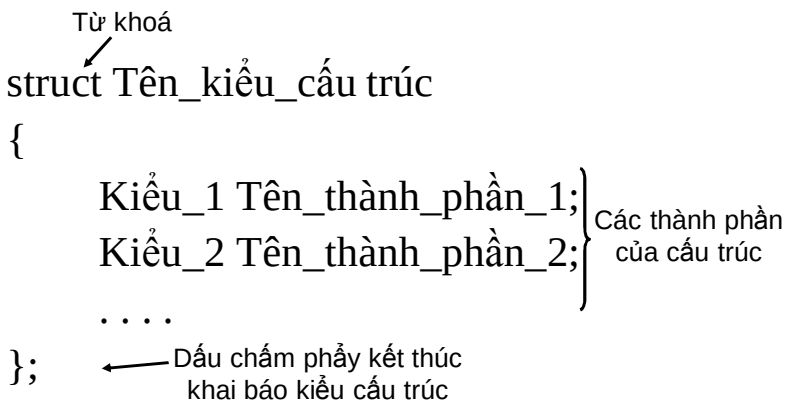
I. Kiểu cấu trúc

1. Khái niệm về kiểu cấu trúc

- Ngoài các kiểu dữ liệu có sẵn trong C, người lập trình còn có thể tạo ra những kiểu dữ liệu của riêng mình: Kiểu cấu trúc và kiểu liệt kê.
- Một cấu trúc là một nhóm các phần tử có thể có kiểu dữ liệu khác nhau. Các phần tử này gọi là các thành phần của cấu trúc. Kiểu cấu trúc trong C tương đương với kiểu bản ghi trong Pascal.

2. Khai báo kiểu cấu trúc

- Khai báo cấu trúc là mô tả về các thành phần của cấu trúc. Cú pháp như sau:



2. Khai báo kiểu cấu trúc (tiếp)

- Ví dụ: Để lưu trữ thông tin về nhân sự của phòng tổ chức với các thông tin về họ tên, ngày sinh, địa chỉ, lương ta khai báo một kiểu cấu trúc như sau:

```
struct nhansu
 {
    char hoten[30];
    char ngaysinh[10];
    char diachi[40];
    float luong;
 };
```


2. Khai báo kiểu cấu trúc (tiếp)

- Sau khi khai báo kiểu cấu trúc ta có thể dùng tên kiểu cấu trúc như tên các kiểu dữ liệu cơ bản.
- Kiểu của các thành phần của cấu trúc có thể là kiểu cấu trúc, tức là trong cấu trúc có thể chứa cấu trúc khác. Ví dụ:

```
struct ngaythang
{
     int ngay,thang,nam;
};
struct nhansu
{
    char hoten[30];
    ngaythang ngaysinh;
    char diachi[40];
    float luong;
};
```


3. Khai báo biến cấu trúc

- Việc khai báo kiểu cấu trúc không tạo ra vùng nhớ chứa cấu trúc mà chỉ mô tả về cấu trúc xem có những gì.
- Muốn có vùng nhớ chứa cấu trúc ta phải khai báo biến cấu trúc. Cú pháp:

  struct Tên_kiểu_cấu_trúc Tên_biến_cấu trúc;

Ví dụ:

```
struct nhansu  ng1,ng2;
```

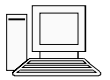

4. Truy nhập các thành phần cấu trúc

- Để truy nhập các thành phần của cấu trúc ta dùng toán tử chấm. Cú pháp:

  Tên_biến_cấu_trúc.Tên_thành_phần

Ví dụ:

```
struct thisinh
  {
    char SBD[15];
    float toan,ly,hoa;
  } ts;
 //Khai bao bien cau truc
 thisinh ts;
 //Nhap du lieu cho thi sinh
 printf("So bao danh: "); scanf("%s",&ts.SBD);
 printf("Diem Toan: "); scanf("%f",&ts.toan);
 printf("Diem Ly: "); scanf("%f",&ts.ly);
 printf("Diem Hoa: "); scanf("%f",&ts.hoa);
```



5. Khởi tạo biến cấu trúc

- Khi khai báo biến cấu trúc ta có thể khởi tạo giá trị cho các thành phần của cấu trúc như khởi tạo cho các phần tử của mảng.

Ví dụ:

```
//Khai bao kieu cau truc
 struct thisinh
   {
    char SBD[15];
    float toan,ly,hoa;
   };
 //Khai bao va khoi tao bien cau truc
 struct thisinh ts={"NNHA23456", 7, 8, 9};
```


6. Phép gán biến cấu trúc

- Ta có thể gán một biến cấu trúc cho một biến cấu trúc cùng kiểu. Ví dụ:

```
//Khai bao kieu cau truc
 struct thisinh
 {
    char SBD[15];
    float toan,ly,hoa;
  };
 //Khai bao bien cau truc
 struct thisinh ts1={"NNHA23456",7,8,9};
 struct thisinh ts2;
 ts2=ts1;
```


7. Mảng cấu trúc

- Sau khi khai báo kiểu cấu trúc thì tên kiểu cấu trúc được dùng như các kiểu dữ liệu khác. Chẳng hạn, dùng cấu trúc làm kiểu phần tử của mảng.

Ví dụ:

```
//Khai bao kieu cau truc
 struct thisinh
   {
    char SBD[15];
    float toan,ly,hoa;
   };
 //Khai bao bien cau truc
 thisinh ds[100];
 strcpy(ds[0].SBD,"NNHA23456");
 ds[0].toan=8;
 ds[0].ly=8;
 ds[0].hoa=9;
```


Ví dụ

Viết chương trình quản lý điểm môn học của sinh viên. Mỗi sinh viên có các thông tin về họ tên, lớp, điểm kiểm tra 1, điểm kiểm tra 2, trung bình kiểm tra, điểm thi, điểm môn học. Nhập vào một danh sách n sinh viên. Hiện danh sách ra màn hình theo dạng bảng với các cột STT, Họ và tên, Lớp, TB KTra, Đ.Thi, Điểm MH. Khi nhập dữ liệu chỉ nhập Họ tên, Lớp, Điểm Ktra1, Điểm Ktra2, còn TB KTra = (Điểm KTra1 + Điểm Ktra2)/2, Điểm MH = 0,3xTB KTra + 0,7xĐiểm Thi.

II. Kiểu liệt kê

- Kiểu liệt kê là kiểu dữ liệu do người lập trình tự định nghĩa bằng cách liệt kê tất cả các giá trị. Các giá trị của kiểu liệt kê là các tên tự đặt.
- Để định nghĩa kiểu liệt kê ta dùng từ khóa enum theo cú pháp sau:

  enum Tên_kiểu_liệt_kê {Danh sách các tên tự đặt};

  Ví dụ: enum boolean {TRUE, FALSE};

  enum mausac {Xanh, Do, Tim, Vang};

  enum days_of_week {Sun, Mon, Tue, Wed, Thu, Fri, Sat};

II. Kiểu liệt kê (tiếp)

- Sau khi khai báo kiểu liệt kê ta có thể khai báo các biến kiểu liệt như các biến kiểu khác:

  Tên_kiểu_liệt_kê Danh_sách_các_biến;

  Ví dụ: Giả sử các kiểu liệt kê đã được khai báo ở trên, ta khai báo các biến liệt kê:

  days_of_week day1, day2;
- Để đưa giá trị vào biến liệt kê ta dùng lệnh gán:

  Ví dụ: day1 = Mon; day2 = Sat;

II. Kiểu liệt kê (tiếp)

- Các giá trị kiểu liệt kê được lưu trữ như các số nguyên kiểu int, giá trị tên đầu tiên là 0, giá trị tên tiếp theo là 1,…

  Ví dụ: Với kiểu liệt kê days_of_week ở trên thì Sun có giá trị 0, Mon có giá trị 1, Tue có giá trị 3,…
- Ta có thể thay đổi giá trị số của các giá trị tên
  - Cho các giá trị tên có giá trị số bắt đầu từ một số khác 0

    Ví dụ: enum mausac {Xanh=5, Do, Tim, Vang};

    Với khai báo này Xanh có giá trị 5, Do có giá trị 6, Tim có giá trị 7, Vàng có giá trị 8.

Chương 7. Con trỏ

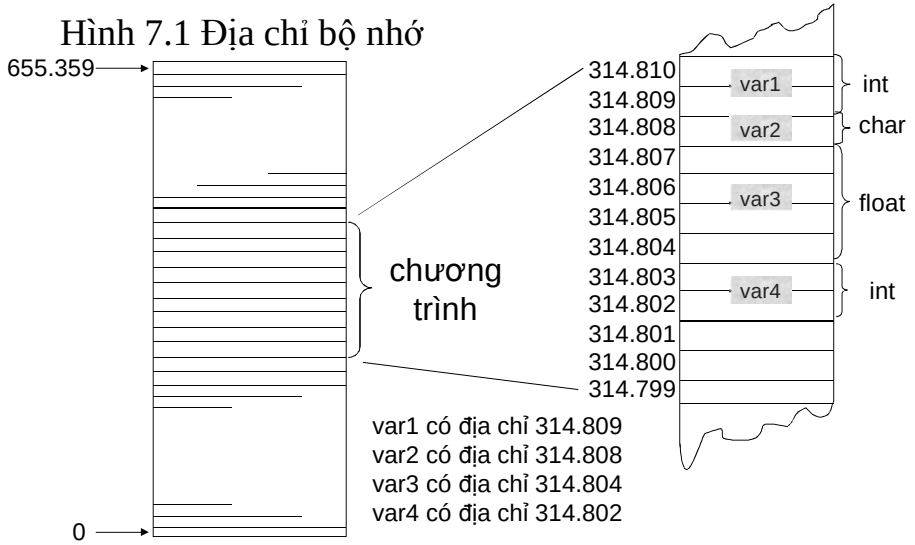
I. Địa chỉ và con trỏ

1. Địa chỉ (hằng con trỏ)

- Mỗi byte trong bộ nhớ máy tính có một địa chỉ. Các địa chỉ này là các số bắt đầu từ 0 trở đi. Ví dụ có 1 MB bộ nhớ thì địa chỉ thấp nhất là 0 và địa chỉ cao nhất là 1.048.575.
- Bất kỳ chương trình nào khi được nạp vào bộ nhớ đều chiếm một khoảng địa chỉ. Điều đó có nghĩa là mọi biến và mọi hàm trong chương trình đều bắt đầu tại một địa chỉ cụ thể. Hình 7.1 cho thấy các địa chỉ bộ nhớ.

1. Địa chỉ (hằng con trỏ) *tiếp*

Hình 7.1 Địa chỉ bộ nhớ



2. Toán tử địa chỉ &

- Toán tử địa chỉ ký hiệu là &, được dùng để lấy địa chỉ của một biến. Toán tử & phải đặt trước tên biến muốn lấy địa chỉ. Ví dụ: Chương trình sau sẽ đưa ra địa chỉ của 3 biến nguyên a, b, c.

3. Khai báo biến con trỏ

- Vì địa chỉ bộ nhớ là số nên nó cũng có thể lưu trữ trong một biến giống như giá trị của các kiểu int, char và float. Một biến mà chứa giá trị địa chỉ gọi là biến con trỏ hay gọi tắt là con trỏ. Nếu một con trỏ chứa địa chỉ của một biến thì ta nói rằng con trỏ trỏ tới biến đó.
- Để khai báo các biến con trỏ ta dùng cú pháp sau:

  Kiểu Tên_biến_con_trỏ;*

  trong đó *Kiểu* là kiểu dữ liệu của đối tượng mà biến con trỏ sẽ trỏ tới. Dấu * có nghĩa là trỏ tới. Nên để dấu * bên cạnh tên kiểu để nhấn mạnh rằng nó là một phần của kiểu chứ không phải của tên biến con trỏ.

3. Khai báo biến con trỏ (tiếp)

- Ví dụ:

  int a;

  int* ptr;

  ptr = &a;

  Lệnh này khai báo một biến con trỏ có tên là ptr trỏ tới các số nguyên int. Nói cách khác con trỏ ptr có thể chứa địa chỉ của các biến nguyên.
- Để khai báo nhiều biến con trỏ cùng trỏ tới một kiểu dữ liệu ta viết:

  *Kiểu *Biến1, *Biến2, *Biến3,…;*

  Mặc dù dấu * để cạnh tên biến con trỏ nhưng vẫn nên hiểu nó là một phần của kiểu.

  Ví dụ: int *p, *q;

3. Khai báo biến con trỏ (tiếp)

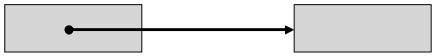
- Khi khai báo một biến con trỏ thì biến con trỏ này sẽ chứa một giá trị vô nghĩa (trừ khi được khởi tạo). Giá trị vô nghĩa này có thể là địa chỉ của một ô nhớ nào đó nằm trong phần chương trình của ta hoặc hệ điều hành. Điều này sẽ rất nguy hiểm nếu ta đưa giá trị vào ô nhớ do con trỏ này trỏ tới. Bởi vậy, trước khi sử dụng một con trỏ ta phải đưa địa chỉ vào nó.
- Con trỏ trỏ tới kiểu nào thì chỉ chứa được địa chỉ của các biến kiểu đó. Không thể gán địa chỉ của biến float tới một con trỏ trỏ tới int.

4. Truy nhập biến qua con trỏ

- Một câu hỏi đặt ra là nếu không biết tên một biến mà chỉ biết địa chỉ của nó thì có truy nhập được vào biến đó không? Câu trả lời là có. Con trỏ chứa địa chỉ của một biến nên ta có thể truy nhập biến qua con trỏ.
- Để truy nhập tới biến do con trỏ ptr trỏ tới ta dùng toán tử truy nhập gián tiếp * đặt trước tên biến con trỏ: *ptr. *ptr tương đương với tên của biến, chỗ nào dùng được tên biến thì chỗ đó dùng được *ptr.

4. Truy nhập biến qua con trỏ

- Toán tử truy nhập gián tiếp cũng ký hiệu là * nhưng có nghĩa là *giá trị của biến được trỏ tới bởi biến con trỏ nằm bên phải nó*, khác với dấu * khi khai báo biến con trỏ có nghĩa là *trỏ tới*.
- Ví dụ:




```
int v;      //Khai báo biến có kiểu int
int* p;     //Khai báo biến con trỏ p trỏ tới int
p = &v;     //Gán địa chỉ của biến v cho con trỏ p
v = 3;      //Gán 3 vào v
*p = 3;     //Gán 3 vào v gián tiếp qua con trỏ p
```


5. Con trỏ trỏ tới **void** và con trỏ NULL

- Ta biết rằng con trỏ trỏ tới kiểu nào thì chỉ chứa được địa chỉ của các biến kiểu đó. Tuy nhiên trong C++ còn có một loại con trỏ đa năng có thể trỏ tới bất kỳ kiểu dữ liệu nào. Con trỏ đó gọi là con trỏ trỏ tới void. Khai báo con trỏ trỏ tới void như sau:

```
void* ptr;
```

- Con trỏ NULL là con trỏ không trỏ tới bất cứ cái gì, nó chứa giá trị rỗng (bằng 0). Để có con trỏ rỗng ta gán giá trị 0 vào biến con trỏ. Ta có thể sử dụng tên hằng này để tạo con trỏ rỗng.

```
int* ptr=NULL;
```


5. Con trỏ trỏ tới **void** và con trỏ NULL (tiếp)

- **Ví dụ:**

```
int ivar;
float fvar;
int* iptr;
float* fptr;
void* vptr;
iptr = &ivar;
//iptr = &fvar;  //lỗi vì gán float*  tới  int*
fptr = &fvar;
//fptr = &ivar;  //lỗi vì gán int*  tới  float*
vptr = &ivar;   //được vì gán int*  tới  void*
vptr = &fvar;   //được vì gán float*  tới  void*
```

6. Các phép toán trên con trỏ

- **Các phép toán số học:**
  - Chỉ có 4 phép toán dùng được với con trỏ là +, -, ++, --.
  - Khi cộng hoặc trừ biến con trỏ với một số thì số đó phải nguyên.
  - Các phép toán số học tác động trên con trỏ khác với bình thường. Cụ thể là khi tăng biến con trỏ lên 1 đơn vị thì địa chỉ chứa trong biến con trỏ không tăng lên một mà tăng lên một lượng bằng kích thước kiểu dữ liệu con trỏ trỏ tới (thường là 2 với kiểu int, 4 với kiểu float và 8 với kiểu double).

6. Các phép toán trên con trỏ (tiếp)

- Ví dụ: giả sử p là con trỏ int chứa địa chỉ 200, sau khi lệnh

  ++p;

  được thực hiện thì p sẽ có giá trị là 202. Nếu p là con trỏ float thì sau lệnh trên p sẽ có giá trị là 204.

- **Các phép toán so sánh:** có thể so sánh hai biến con trỏ bằng các phép toán so sánh. Tuy nhiên việc so sánh này chỉ có ý nghĩa trong hai trường hợp sau:
  - So sánh hai con trỏ để xem chúng có bằng con trỏ NULL không.

6. Các phép toán trên con trỏ (tiếp)

  - So sánh hai con trỏ khi chúng cùng liên quan tới một đối tượng, chẳng hạn là cùng trỏ tới một biến.
- **Phép gán:** Có thể gán một biến con trỏ cho một biến con trỏ có cùng kiểu trỏ tới.
- **Lưu ý**: Khi dùng toán tử tăng hoặc giảm với biến do con trỏ trỏ tới thì phải chú ý về thứ tự thực hiện các phép toán. Ví dụ: nếu ta viết

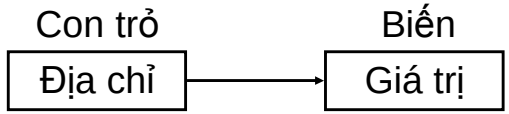
  *p++;

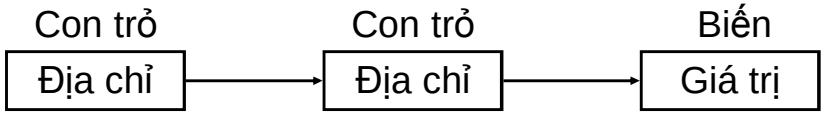
  thì con trỏ sẽ tăng lên 1 chứ không phải biến do con trỏ trỏ tới tăng lên 1, bởi vì phép toán * và ++ cùng mức ưu tiên, được kết hợp từ phải qua trái. Muốn tăng biến do con trỏ trỏ tới ta phải viết:

  (*p)++;

7. Con trỏ trỏ tới con trỏ

- Trong C++, một con trỏ có thể trỏ tới một con trỏ khác, tức là một con trỏ có thể chứa địa chỉ của một biến con trỏ khác.





7. Con trỏ trỏ tới con trỏ (tiếp)

- Để khai báo một biến con trỏ trỏ tới một con trỏ ta dùng thêm dấu * nữa. Ví dụ:

  int** p; //p là con trỏ trỏ tới một con trỏ int
- Để truy nhập tới biến qua con trỏ trỏ tới con trỏ ta phải dùng hai lần toán tử truy nhập gián tiếp. Kiểu truy nhập này gọi là truy nhập gián tiếp bội (Multiple Indirection). Ví dụ:

```
char ch;
char* p;
char** mp;
ch='A';
p=&ch;
mp=&p;
cout<<"Ky tu nam trong bien ch la: "<<**mp;
```

II. Con trỏ, mảng và xâu ký tự

1. Con trỏ và mảng

- Con trỏ được sử dụng để truy nhập vào các phần tử của mảng và làm đối số truyền vào hàm. Và khi mảng làm đối số truyền vào hàm thì con trỏ cũng rất hữu ích.
- Các phần tử của mảng có thể được truy nhập qua ký hiệu của mảng ([]) hoặc ký hiệu của con trỏ (*). Ví dụ:

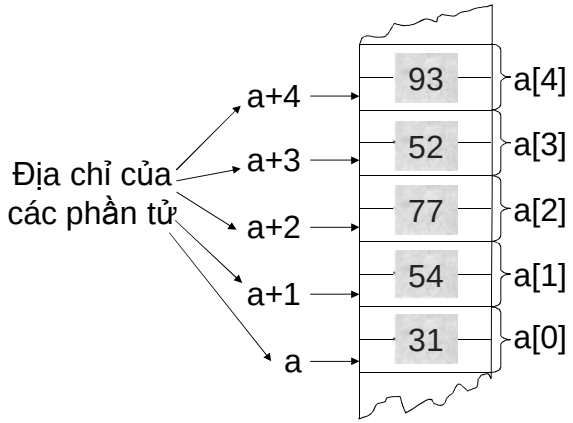
```
int a[5]={31,54,77,52,93};
int i;
//Dua ra bang ky hieu cua mang
for(i=0;i<5;i++) printf("%i ",a[i]);
//Dua ra bang ky hieu cua con tro
for(i=0;i<5;i++) printf("%i ",*(a+i));
```

1. Con trỏ và mảng (tiếp)

- Biểu thức *(a+i) tương đương với a[i]. Ví dụ, với i=2 thì *(a+2) là phần tử thứ 3 (có giá trị là 77).
- Tại sao *(a+2) lại là phần tử thứ 3? Như ta đã biết, tên biến mảng chính là địa chỉ của phần tử đầu tiên của biến mảng. Khi ta viết (a+2) thì trình biên dịch sẽ thực hiện cộng địa chỉ với 2. Khi cộng địa chỉ với 2 trình biên dịch lấy kích thước kiểu dữ liệu của mảng nhân với 2 rồi mới cộng vào địa chỉ. Kết quả (a+2) cho ta địa chỉ của phần tử thứ 3. Để truy nhập tới phần tử thứ 3 khi biết địa chỉ phải sử dụng toán tử truy nhập gián tiếp *(a+2).

1. Con trỏ và mảng (tiếp)

- Địa chỉ của các phần tử mảng



1. Con trỏ và mảng (tiếp)

- **Hằng con trỏ và biến con trỏ:** Tên biến mảng là một địa chỉ cụ thể mà hệ thống đã chọn để đặt mảng. Địa chỉ này không thể thay đổi và nó được duy trì khi biến mảng còn tồn tại. Người ta gọi các địa chỉ không thay đổi được là các hằng con trỏ. Vì tên biến mảng a ở ví dụ trên là hằng nên ta không thể viết a++ hay a+=2.

  Một địa chỉ thì không thể thay đổi nhưng biến con trỏ chứa địa chỉ thì có thể thay đổi.

1. Con trỏ và mảng (tiếp)

- **Hằng con trỏ và biến con trỏ: (tiếp)**

  Ví dụ sau dùng biến con trỏ để đưa ra các phần tử của mảng:

```
int a[5]={31,54,77,52,93};
int i;
int *p=a;  //p tro toi phan tu dau tien cua mang a
//Dua ra bang bien con tro
cout<<"Dua ra bang bien con tro: "<<'\n';
for(i=0;i<5;i++) cout<<*p++<<' ';
```


2. Con trỏ và xâu ký tự

- Như ta đã biết, xâu ký tự thực chất là mảng ký tự. Bởi vậy ta có thể dùng ký hiệu con trỏ để truy nhập vào các ký tự của xâu giống như truy nhập vào các phần tử của mảng. Ví dụ:

  char s[6]=”DHNNI”;

  cout<<*(s+1);//Dua ra ky tu thu 2 la H
- **Con trỏ trỏ tới hằng xâu ký tự:** Khi khai báo và khởi tạo biến xâu ký tự ta có thể khai báo như một mảng ký tự hoặc khai báo như một con trỏ trỏ tới kiểu ký tự. Ví dụ:

  char s1[] = ”Khai bao nhu mot mang”; s1[1], *(s1+1)

  //char* s1 = ”Khai bao nhu con con tro”; *(s1+1), s1[1]

2. Con trỏ và xâu ký tự (tiếp)

Sau khai báo trên ta sẽ được hai biến xâu ký tự s1 và s2. Tuy nhiên hai biến xâu này có một sự khác nhau: s1 là một địa chỉ, một hằng con trỏ, s2 là một biến con trỏ; s2 có thể thay đổi còn s1 không thể thay đổi. Ví dụ:

```
char s1[]="Khai bao nhu mot mang";
char* s2 ="Khai bao nhu mot con tro";
cout<<s1<<'\n';
cout<<s2<<'\n';
//s1++;              //Bao loi, s1 la hang con tro
s2++;                //Duoc
cout<<s2;            //Chi hien: hai bao nhu mot con tro
```

Chú ý: Khi thay đổi s2 thì ký tự đầu tiên của xâu sẽ thay đổi. Ở ví dụ trên, sau khi tăng s2 lên 1 thì ký tự đầu tiên của xâu là h.

2. Con trỏ và xâu ký tự (tiếp)

- **Mảng con trỏ trỏ tới các hằng xâu ký tự:**
  - Giống như mảng các biến kiểu int hoặc float, ta cũng có mảng con trỏ. Mảng con trỏ hay dùng nhất là mảng con trỏ trỏ tới các hằng xâu ký tự.
  - Ta xét hai cách khai báo sau đây:

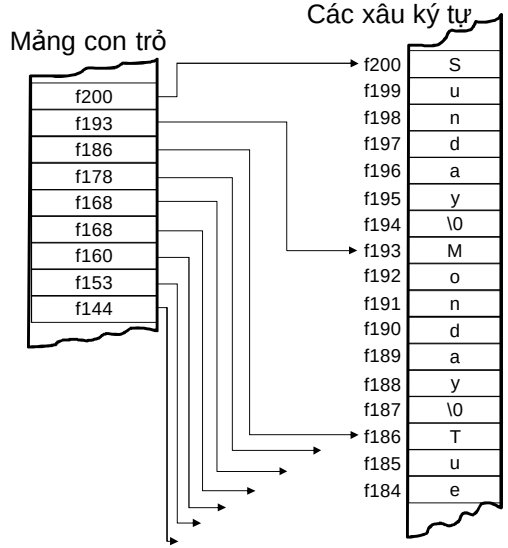
```
//Dùng mảng hai chiều
char days[7][10]={"Sunday","Monday","Tuesday","Wednesday",
                  "Thursday","Friday","Saturday"};
//Dùng con trỏ
char* days[7]={"Sunday","Monday","Tuesday","Wednesday",
               "Thursday","Friday","Saturday"};
```


2. Con trỏ và xâu ký tự (tiếp)

- **Mảng con trỏ trỏ tới các hằng xâu ký tự: (tiếp)**
  - Nếu khai báo theo mảng hai chiều thì các mảng con chứa các xâu ký tự phải có kích thước bằng nhau (10). Do đó, với những xâu có số ký tự nhỏ hơn 10 sẽ gây lãng phí bộ nhớ.
  - Nếu khai báo theo con trỏ thì trình biên dịch C++ sẽ để các xâu ký tự liên tiếp nhau trong bộ nhớ và dùng một mảng con trỏ để trỏ tới các xâu này (Hình trang sau cho thấy các xâu ký tự trong bộ nhớ). Một xâu ký tự là một mảng kiểu char, do đó một mảng con trỏ trỏ tới xâu ký tự thực chất là một mảng con trỏ trỏ tới char. Đây chính là lý do tại sao ta khai báo là char*

2. Con trỏ và xâu ký tự (tiếp)

Địa chỉ của ký tự đầu tiên chính là địa chỉ của xâu. Các địa chỉ này được lưu trữ trong mảng con trỏ.



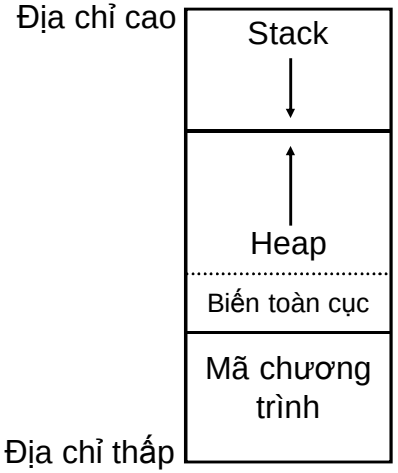
Bài tập

Viết chương trình nhập vào một họ tên. Tách tên và đưa tên ra màn hình.

III. Quản lý bộ nhớ với malloc và free

1. Cách sử dụng bộ nhớ của một chương trình C

- Một chương trình C khi chạy sẽ chiếm một vùng nhớ trong bộ nhớ. Vùng nhớ này được chia thành 3 phần: phần chứa mã chương trình, phần chứa các biến tĩnh và biến ngoài (gọi là Heap), phần chứa các biến tự động (gọi là Stack). Stack mở rộng từ địa chỉ cao xuống địa chỉ thấp, Heap mở rộng từ địa chỉ thấp lên địa chỉ cao.



2. Các loại biến trong chương trình C

a) Sự khác nhau giữa khai báo và định nghĩa

b) Thời gian tồn tại và phạm vi hoạt động của các loại biến

a) Sự khác nhau giữa khai báo và định nghĩa

- Một khai báo (declaration) chỉ xác định tên và kiểu dữ liệu. Nhiệm vụ của khai báo là cung cấp thông tin cho trình biên dịch, nó không yêu cầu trình biên dịch làm bất cứ việc gì.
- Trái lại, một định nghĩa (definition) yêu cầu trình biên dịch phải cấp phát bộ nhớ cho biến.
- Trong một số trường hợp khai báo cũng yêu cầu trình biên dịch cấp phát bộ nhớ, chẳng hạn như khai báo biến. Tuy nhiên, với định nghĩa thì trong bất kỳ trường hợp nào cũng yêu cầu cấp phát bộ nhớ.

b) Thời gian tồn tại và phạm vi hoạt động của các loại biến

- Các loại biến có hai đặc tính chính là phạm vi hoạt động và thời gian tồn tại. Phạm vi hoạt động liên quan đến phần chương trình nào có thể truy nhập (sử dụng) biến. Thời gian tồn tại là khoảng thời gian trong đó biến tồn tại. Phạm vi hoạt động của biến có thể là trong một lớp, một hàm, một file hay một số file. Thời gian tồn tại của một biến có thể trùng với một đối tượng, một hàm hay toàn bộ chương trình.
- Có các loại biến sau: biến tự động, biến thanh ghi, biến trong khối lệnh, biến ngoài, biến tĩnh.

Các biến tự động (automatic variable)

- Các biến tự động là các biến được khai báo trong một hàm. Sở dĩ gọi chúng là các biến tự động bởi vì chúng được tự động tạo khi hàm được gọi và bị hủy khi hàm kết thúc.
  - Biến tự động có phạm vi hoạt động trong một hàm. Do đó, một biến i được khai báo trong một hàm hoàn toàn khác với một biến i được khai báo trong một hàm khác.
  - Mặc định các biến tự động không được khởi tạo, bởi vậy ngay sau khi chúng được hình thành chúng sẽ có một giá trị vô nghĩa.

Các biến thanh ghi (register variable)

- Biến thanh ghi là một loại biến tự động đặc biệt. Nó được đặt trong các thanh ghi của CPU chứ không phải trong bộ nhớ. Việc truy nhập các biến thanh ghi nhanh hơn các biến thông thường. Biến thanh ghi có lợi nhất khi được dùng làm biến điều khiển cho lệnh lặp bên trong nhất trong các lệnh lặp lồng nhau. Ta chỉ nên dùng một đến hai biến thanh ghi trong một hàm.
- Để khai báo biến thanh ghi ta dùng từ khóa register trước khai báo biến thông thường.

  Ví dụ: register int a;

Các biến trong khối lệnh

- Các biến tự động có thể được khai báo ở bất kỳ đâu trong một hàm hoặc trong một khối lệnh. Khối lệnh là phần chương trình nằm giữa hai dấu ngoặc { và }, chẳng hạn như thân lệnh if hay thân lệnh lặp. Các biến được khai báo trong một khối lệnh có phạm vi hoạt động chỉ trong khối lệnh đó.

Các biến ngoài (external variable)

- Các biến ngoài là các biến được khai báo ở bên ngoài tất cả các hàm. Các biến ngoài có phạm vi hoạt động từ vị trí khai báo đến cuối file khai báo chúng. Thời gian tồn tại của các biến ngoài là thời gian tồn tại của chương trình, tức là khi chương trình kết thúc thì các biến ngoài mới bị hủy. Khác với các biến tự động, các biến ngoài được tự động khởi tạo bằng 0 nếu ta không khởi tạo.

Các biến ngoài (*tiếp*)

```
//Bat dau file
int a; //a la bien ngoai
.....
void afunc();
.......
//Cuoi file
```


Các biến ngoài (*tiếp*)

- Nếu chương trình được chia thành nhiều file thì các biến ngoài chỉ có thể dùng được trong file khai báo chúng, không dùng được trong các file khác. Để sử dụng một biến ngoài đã được định nghĩa ở một file thì ta phải khai báo biến đó dùng từ khóa extern.
- Để các biến ngoài chỉ truy nhập được trong file khai báo chúng, không truy nhập được từ file khác ta dùng từ khóa static. Từ khóa static sẽ hạn chế phạm vi hoạt động của biến.

  Ví dụ: (*trang sau*)

Các biến ngoài (*tiếp*)

Ví dụ 1: Truy nhập biến ngoài trên nhiều file

```
//Bat dau file 1
int a;    //a la bien ngoai
//Cuoi file 1

//Bat dau file 2
extern int a;     //khai bao su dung bien ngoai a o file 1
          //Trong file 2 co the truy nhap bien a
//Cuoi file 2

//Bat dau file 3
//Khong khai bao su dung bien ngoai a nen trong file 3
// khong the truy nhap bien a
//Cuoi file 3
```


Các biến ngoài (*tiếp*)

Ví dụ 2: Hạn chế việc truy nhập biến ngoài

```
//Bat dau file 1
static int a;   //dinh nghia bien ngoai a
                //bien a chi truy nhap duoc trong file nay
//Cuoi file 1

//Bat dau file 2
extern int a; //Khong dung duoc khai bao nay
//Cuoi file 2
```


Các biến ngoài (*tiếp*)

- Có hai vấn đề khi sử dụng biến ngoài:
  - Vì biến ngoài có thể truy nhập được từ bất kỳ hàm nào trong chương trình nên rất dễ bị thay đổi làm mất dữ liệu.
  - Vì các biến ngoài có phạm vi hoạt động ở mọi nơi trong chương trình nên ta phải quan tâm đến vấn đề kiểm soát tên biến để sao cho không có hai biến nào trùng tên.

Các biến tĩnh cục bộ (local static)

- Các biến tĩnh cục bộ được sử dụng khi ta muốn duy trì giá trị của một biến khai báo trong hàm giữa các lời gọi hàm. Tức là khi hàm kết thúc biến tĩnh vẫn còn và vẫn chứa giá trị, khi hàm được gọi lần 2 lại có thể sử dụng giá trị này. Phạm vi hoạt động của biến tĩnh cục bộ là trong hàm nhưng thời gian tồn tại của nó là suốt thời gian chương trình chạy.

3. Hạn chế của việc lưu trữ bằng mảng

- Mảng rất hay được sử dụng khi cần lưu trữ một số lượng lớn các biến hay đối tượng. Tuy nhiên tại thời điểm viết chương trình ta phải xác định kích thước của mảng chứ không đợi được đến khi chương trình thực hiện. Đoạn chương trình sau sẽ sinh ra lỗi:

```
printf("Nhap vao kich thuoc mang: ");scanf("%i",&size);
int a[size];        //Lỗi, kích thước mảng phải là hằng
```

- Trong nhiều trường hợp, tại thời điểm viết chương trình ta không biết được là cần bao nhiêu bộ nhớ. Nếu dự trù nhiều mà không dùng hết thì lãng phí bộ nhớ, nếu dự trù ít mà cần lưu trữ nhiều thì không có chỗ chứa. Vấn đề này được khắc phục bằng cơ chế cấp phát động bộ nhớ hàm malloc() và free().

4. Hàm malloc() và free()

- Trong C có 2 hàm thực hiện chức năng cấp phát và giải phóng bộ nhớ, đó là hàm malloc() và free(). Muốn sử dụng hai hàm này trong chương trình ta phải khai báo sử dụng thư viện stdlib (#include<stdlib.h>).
- Cú pháp cấp phát bộ nhớ động như sau:

```
Biến_con_trỏ = (Kiểu_dl_của_biến*) malloc(sizeof(Kiểu_dl_của_biến));
```

trong đó Biến con trỏ phải được khai báo trỏ đến kiểu dữ liệu của biến.

Ví dụ: int* p=NULL;

```
p = (int*) malloc(sizeof(int));
```


4. Hàm malloc() và free()

- Hàm malloc() sẽ cấp phát một ô nhớ trong phần nhớ Heap trong khi chương trình đang chạy, đủ để chứa một giá trị có kiểu Kiểu_dl_của_biến và trả về một con trỏ trỏ tới nó.
- Vì kích thước phần Heap có giới hạn nên có thể sẽ hết. Nếu phần nhớ Heap đã hết mà ta vẫn cấp phát thì hàm malloc() sẽ trả về con trỏ rỗng (NULL). Bởi vậy, luôn luôn phải kiểm tra con trỏ được trả về bởi hàm malloc() trước khi dùng nó.

```
if(!Biến_con_trỏ) printf("Cap phat bo nho bi loi!");
```


4. Hàm malloc() và free() (*tiếp*)

- Cú pháp giải phóng bộ nhớ được cấp phát bởi hàm malloc()

```
free(p);
```

Trong đó p là con trỏ trỏ tới vùng nhớ được cấp phát động bởi hàm malloc().

- Hàm free(p) sẽ giải phóng vùng nhớ được trỏ tới bởi biến con trỏ p. Chỉ nên dùng hàm free() để giải phóng vùng nhớ được cấp phát bởi hàm malloc(), calloc() hoặc realloc().

4. Hàm malloc() và free() (*tiếp*)

- Ví dụ về sử dụng hàm malloc() và free():

```
//Khai bao su dung thu vien chuong trinh
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int* p;
  p = (int*) malloc(sizeof(int));              //cap phat bo nho chua kieu int
  if(p != NULL)
    {
      printf("Cap phat bo nho bi loi“);
      return 1;
    }
  *p=100; //Gan 100 vao o nho vua duoc cap
  printf(”*p = %i”,*p);            //Hien thi noi dung cua o nho vua duoc cap
  free(p);  //Giai phong o nho vua duoc cap
return 0;
}
```


5. Cấp phát mảng động với calloc()

- Với cơ chế cấp phát động bộ nhớ ta có thể cấp phát bộ nhớ cho cả một biến mảng. Điều này cho phép xác định số phần tử của mảng trong khi chạy chương trình. Để cấp phát động cho mảng một chiều ta dùng hàm calloc() với cú pháp như sau:

  Biến_con_trỏ = (KPT*) calloc(size, sizeof(KPT));

  trong đó KTP là kiểu phần tử, size là số phần tử của mảng. size có thể là hằng, biến hoặc biểu thức.

  int a[100];

  int *a = (int*) calloc(100,sizeof(int)); a[0]=25; a[1]=15;

- Để giải phóng vùng nhớ cấp phát cho mảng ta dùng hàm free() với đối số là Biến con trỏ trỏ tới mảng:

  free(Biến_con_trỏ);

Ví dụ

Nhập vào dãy số nguyên có n phần tử. Tìm phần tử có giá trị lớn nhất.

5. Mảng động (*tiếp*)

- Với mảng động ta có thể thay đổi kích thước của mảng mà vẫn giữ được nội dung của mảng ban đầu.
- Để thay đổi kích thước một mảng động ta dùng hàm realloc() với cú pháp sau:

  p = (KPT*) realloc(p, size_new);

  Trong đó KPT là kiểu phần tử, p là con trỏ trỏ tới mảng động, size_new là kích thước mới của mảng động.

  Nếu thay đổi kích thước không thành công, hàm realloc() sẽ trả về con trỏ rỗng (NULL).

Ví dụ

Nhập vào dãy số nguyên có n phần tử. Chèn thêm phần tử x vào cuối dãy.

5. Mảng động (tiếp)

- Ví dụ về mảng động:

```
//Khai bao su dung thu vien chuong trinh
#include<stdio.h>
#include<stdlib.h>
int main()
{
  int* a;
  int n,i;

  printf("Nhap vao so phan tu cua mang: ");scanf("%i",&n);
  a = (int*) calloc(n, sizeof(int));//Cap phat bo nho cho mang n phan tu nguyen
  if(!a)
    {
      printf("Cap phat bo nho bi loi");
      return 1;
    }
```

//Tiếp trang sau

5. Mảng động (tiếp)

- Ví dụ về mảng động: (tiếp)

```
//Nhap cac gia tri vao mang
 printf("Nhap vao mang so nguyen:\n" );
 for(i=0;i<n;i++)
   {
     printf("Nhap vao so thu %i: ",i+1); scanf("%i",&a[i]);
   }
 //Mo rong kich thuoc mang them 10 phan tu
 a = (int*) realloc(a, n + 10);
 //Dua cac so nhap vao ra man hinh
 printf("Cac so da nhap la:\n");
 for(i=0;i<n;i++) printf("%i ",a[i]);
 free(a); //Giai phong vung nho cap phat cho mang
 return 0;
}
```


Bài tập chương 7

- Bai 1. Cho dãy số nguyên a_1, a_2, a_3,…, a_n. Sắp xếp dãy số tăng dần. Yêu cầu trong chương trình có sử dụng mảng động để chứa dãy số.
- Bài 2. Viết chương trình nhập vào một dãy n số nguyên, lưu dãy số này trong một danh sách liên kết đơn P. Hãy tạo một danh sách liên kết đơn Q là đảo ngược của P.
- Bài 3. Viết chương trình nhập vào một dãy n số nguyên, lưu dãy số này trong một danh sách liên kết đơn P. Hãy sắp xếp dãy số theo chiều không giảm sử dụng phương pháp sắp xếp chọn.

Chương 8. Hàm trong C

I. Khai báo hàm

1. Giới thiệu về hàm

- Trong C tất cả các chương trình con đều gọi là hàm.
- Ngoài các hàm thư viện có sẵn, người lập trình có thể tự tạo ra các hàm. Để tạo ra một hàm người lập trình phải khai báo và định nghĩa nó.
- Khai báo hàm (function declaration or prototype) là xác định tên của hàm, kiểu dữ liệu trả về, số lượng tham số và kiểu của từng tham số.
- Định nghĩa hàm (function definition) là xác định công việc mà hàm sẽ thực hiện thông qua các lệnh của hàm.
- Các hàm trong C không lồng nhau, tức là trong một hàm ta không thể định nghĩa một hàm khác.

2. Cú pháp khai báo hàm

- Cú pháp khai báo hàm nằm trên một dòng, kết thúc bằng dấu chấm phẩy.

Kiểu_trả_về Tên_hàm(Kiểu_1 Tên_tham_số_1, Kiểu_2 Tên_tham_số_2,…);

Ví dụ: float inchtomet(float x);

int cong(int a, int b);

- Một khai báo hàm không cho biết những gì có trong thân hàm. Nó chỉ báo cho trình biên dịch biết về tên hàm, kiểu của hàm, số lượng các tham số và kiểu của các tham số.

2. Cú pháp khai báo hàm (tiếp)

- Khai báo hàm có thể đặt ở bất kỳ đâu trước khi gọi hàm. Tốt nhất là để ở đầu tệp chứa chương trình chính (chứa hàm main) hoặc để trước một hàm sẽ gọi nó. Trong các chương trình nhiều file thì các khai báo hàm thường để trong các file header có đuôi .h, còn các định nghĩa hàm để trong các file thư viện có đuôi obj hoặc lib.
- Nếu hàm được định nghĩa ở đâu đó trước khi gọi hàm thì có thể không cần khai báo hàm. Tuy nhiên vẫn nên có khai báo hàm nhất là trong các chương trình có nhiều hàm lớn hay các chương trình nằm trên nhiều file.

3. Các tham số trong khai báo hàm

- Nếu hàm không có tham số thì trong dấu ngoặc đơn của khai báo hàm để trống. Ví dụ:

  int xoa();
- Tên của các tham số trong khai báo hàm có thể không cần xác định. Ví dụ:

  float inchtomet(float, float);

II. Định nghĩa hàm

1. Cú pháp định nghĩa hàm

Kiểu_trả_về Tên_hàm(Kiểu_1 Tên_tham_số_1, Kiểu_2 Tên_tham_số_2,…)

{

//Các lệnh của hàm để đây } Thân hàm

}

Không có dấu chấm phẩy

Ví dụ:

```
int cong(int a, int b)
{
    int z;
    z = a + b;
    return z;
}
```

1. Cú pháp định nghĩa hàm (tiếp)

- Dòng đầu tiên trong định nghĩa hàm giống trong khai báo hàm, chỉ khác là không có dấu chấm phẩy và các tham số bắt buộc phải có tên.
- Khi đã có khai báo hàm thì định nghĩa hàm thường để sau hàm main hoặc để trong một tệp obj (lib). Để quen dần với việc viết các chương trình lớn, khi thực hành chúng ta viết các khai báo hàm trong tệp .h, còn các định nghĩa hàm để trong tệp .obj (lib).

2. Lệnh return

- Lệnh return được sử dụng trong một hàm. Lệnh return thực hiện hai chức năng:
  - Làm cho một hàm trở về chương trình gọi nó.
  - Được dùng để trả về một giá trị.
- Cú pháp dùng lệnh return như sau:

  return Giá_trị_trả_về;

  hoặc return;
- Lệnh return có thể dùng ở bất kỳ vị trí nào trong hàm nhưng thường ở cuối hàm.
- Với các hàm có trả về giá trị thì lệnh return bắt buộc phải có.

3. Hàm không trả về giá trị

- Với các hàm không trả về giá trị thì khi khai báo và định nghĩa hàm ta phải khai báo kiểu trả về là void. Ví dụ:

  void chao();
- Nếu sử dụng lệnh return trong hàm không trả về giá trị thì chỉ dùng được dạng:

  return;

III. Sử dụng hàm

1. Lời gọi hàm

- Một hàm, sau khi được định nghĩa và khai báo, có thể được thực hiện bằng một lệnh gọi hàm (lời gọi hàm) ở đâu đó trong chương trình. Có thể gọi từ hàm main, có thể gọi từ một hàm khác hoặc có thể gọi từ một hàm thành viên của lớp.
- Cú pháp gọi hàm như sau:

  Tên_hàm(Danh sách các đối số, nếu có);
- Nếu hàm được khai báo và định nghĩa là có các tham số thì khi gọi hàm ta phải truyền giá trị cho hàm qua các tham số. Các giá trị truyền cho hàm gọi là các đối số. Các đối số có thể là hằng, biến, mảng, con trỏ,…

1. Lời gọi hàm (tiếp)

- Ví dụ: giả sử ta khai báo một hàm cộng hai giá trị float

  int cong(int a, int b);

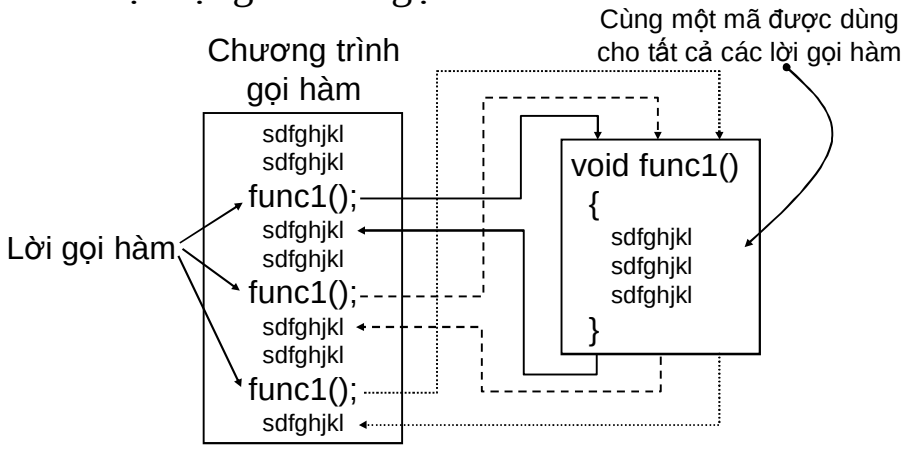
  Ta gọi hàm này như sau:

  cong(7,8);
- Lời gọi một hàm có trả về giá trị có thể sử dụng trong các biểu thức, còn lời gọi một hàm không trả về giá trị không dùng được trong biểu thức. Khi dùng trong biểu thức thì không có dấu chấm phẩy sau lời gọi hàm. Ví dụ:

  a = cong(7,8) +2; cout<<a;

1. Lời gọi hàm (tiếp)

- Hoạt động của lời gọi hàm



Ví dụ về hàm

- Viết chương trình tính số các chỉnh hợp chập k từ n phần tử. Chương trình phải sử dụng hàm để tính giai thừa.

$$A_n^k = \frac{n!}{(n-k)!}$$

2. Truyền đối số theo giá trị (tiếp)

- Trong C, các đối số truyền vào cho hàm là truyền theo giá trị.
- Khi truyền đối số cho hàm theo giá trị thì hàm sẽ tạo ra các biến mới (tên các biến này là tên của các tham số), copy giá trị của các đối số vào các biến mới và thao tác trên các biến mới này. Bởi vậy sau khi gọi hàm các đối số không bị thay đổi giá trị mặc dù bên trong hàm giá trị của đối số bị thay đổi.

2. Truyền đối số theo giá trị (tiếp)

```
#include<iostream.h>
void DoiCho(int a, int b);
int main()
{
   int x=12,y=15;
   clrscr();
   printf("Truoc khi doi cho: x= %d,  y= %d“,x,y);
   DoiCho(x,y);
   printf(“Sau khi doi cho: x= %d,  y= %d“,x,y);
}
void DoiCho(int a,int b)  //Khai bao de truyen doi so theo gia tri
{
   int tmp=a;
   a=b;
   b=tmp;
}
```

3. Truyền con trỏ tới hàm

- Để truyền con trỏ tới hàm ta phải thực hiện hai bước:
  - Khai báo các tham số (khi khai báo và định nghĩa) là con trỏ.
  - Khi gọi hàm thì đối số truyền cho hàm là địa chỉ.

Ví dụ:

```
void DoiCho(int* a, int* b);
int x = 12,y = 15;
DoiCho(&x,&y);
```


3. Truyền con trỏ tới hàm (tiếp)

- Khi truyền con trỏ tới hàm thì biến do con trỏ trỏ tới có thể bị thay đổi bởi hàm.

  Ví dụ: Đổi chỗ giá trị của hai biến

```
void DoiCho(int* a,int* b);
. . . .
DoiCho(&x,&y);
. . . .
void DoiCho(int* a,int* b)
 {
  int tmp = *a;
  *a = *b;
  *b = tmp;
 }
```

4. Truyền mảng tới hàm

- Khi tên mảng được sử dụng mà không có chỉ số kèm theo thì nó là địa chỉ bắt đầu của mảng. Do đó, nếu dùng mảng làm đối số truyền tới một hàm thì chỉ có địa chỉ của mảng được truyền tới hàm chứ không phải toàn bộ mảng. Điều này có nghĩa rằng khi khai báo tham số của hàm thì tham số phải có kiểu con trỏ.
- Bởi vì địa chỉ của mảng được truyền tới hàm nên mọi thay đổi của hàm lên mảng sẽ giữ nguyên khi hàm kết thúc.

4. Truyền mảng tới hàm (tiếp)

- *Ví dụ:* Viết một hàm đưa ra các phần tử của mảng

```
void print(int* m);
. . . . .
int x[7]={2,5,8,1,6,7,10};
. . . . .
print(x);
. . . . .
void print(int* m)
  {
    for(int i=0;i<7;i++) printf("%d  ",m[i]);
  }
```

IV. Con trỏ trỏ tới hàm

- Một đặc điểm rất mạnh của C là con trỏ hàm. Mặc dù hàm không phải là biến nhưng nó vẫn có địa chỉ trong bộ nhớ. Địa chỉ này có thể chứa trong một con trỏ. Vì địa chỉ của hàm chứa trong con trỏ nên ta có thể sử dụng con trỏ thay cho tên hàm.
- Để có địa chỉ của hàm ta dùng tên hàm không có đối số (giống như tên biến mảng là địa chỉ của mảng).

IV. Con trỏ trỏ tới hàm (tiếp)

- Để có con trỏ có thể chứa địa chỉ của hàm ta khai báo con trỏ trỏ tới kiểu giống kiểu trả về của hàm, theo sau là các tham số của hàm đặt trong ngoặc đơn. Ví dụ: giả sử hàm Tong có hai tham số kiểu int, kiểu trả về cũng là int. Khi đó ta khai báo con trỏ trỏ tới hàm này như sau:

```
int Tong(int a, int b); //Khai bao hàm
int (*p) (int a, int b);  //Khai báo con trỏ hàm
p = Tong;  //Gán địa chỉ của hàm Tong cho p
(*p)(10,15);  //Gọi hàm Tong qua con trỏ
```

IV. Con trỏ trỏ tới hàm (tiếp)

- Ví dụ: Viết chương trình tính tổng, hiệu, tích và thương của hai số nguyên nhập vào từ bàn phím. Chương trình yêu cầu người sử dụng lựa chọn một trong các cách tính.

V-Xây dựng thư viện hàm

1) Các khai báo hàm để trong tệp header (.h)
2) Các định nghĩa hàm để trong tệp .c
3) Biên dịch tệp định nghĩa hàm thành tệp .obj
4) Tạo thư viện hàm có đuôi .lib bằng trình tlib
5) Khai báo thư viện hàm trong tệp chương trình chính: #include”thuvien.h”

   Chú ý: Tệp header và tệp thư viện hàm để trong cùng thư mục với tệp chương trình chính.

Chương 9. Kiểu dữ liệu tệp

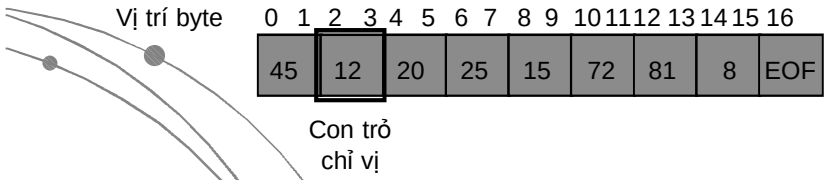
I. Giới thiệu về tệp

1. Khái niệm về tệp

- Kiểu tệp bao gồm một tập hữu hạn các phần tử có cùng kiểu dữ liệu.
- Số phần tử của tệp không cần xác định khi khai báo biến tệp.
- Các phần tử của tệp được lưu trữ trên bộ nhớ ngoài. Đây là đặc điểm khác với tất cả các kiểu dữ liệu khác.

2. Cấu trúc của tệp

- Các phần tử của tệp được sắp xếp thành một dãy các byte liên tiếp nhau. Sau phần tử dữ liệu cuối cùng là phần tử EOF. Phần tử này không phải là dữ liệu mà là mã kết thúc tệp.



3. Phân loại tệp

- Dựa vào cách lưu trữ dữ liệu trên tệp ta có các loại tệp sau:
  - Tệp nhị phân (binary): Dữ liệu ghi ra tệp nhị phân có dạng các byte nhị phân giống như trong bộ nhớ.
  - Tệp văn bản (text): Dữ liệu được ghi ra tệp thành các ký tự trong bảng mã ASCII. Trên tệp văn bản có mã xuống dòng gồm 2 ký tự LF (mã 10) và CR (mã 13).

4. Khai báo tệp

- Kiểu tệp đã được trình biên dịch định nghĩa với tên chuẩn là FILE.
- Khai báo tệp ta khai báo biến con trỏ trỏ tới kiểu FILE.

  Ví dụ: FILE *f;
- Con trỏ tệp sẽ trỏ tới vùng nhớ chứa các thông tin về tệp trên bộ nhớ ngoài.

II. Tệp nhị phân

1. Ghi dữ liệu ra tệp nhị phân

- B1: Mở tệp để ghi bằng hàm fopen()

  fp = fopen(Tên tệp, Kiểu truy nhập);

  trong đó: +) fp là con trỏ tệp được khai báo trỏ tới kiểu FILE;

  +) Tên tệp có thể là hằng xâu hoặc biến xâu. Trong tên tệp có thể có đường dẫn.

  + Kiểu truy nhập tệp là hằng xâu diễn tả cách truy nhập vào tệp.

Các kiểu truy nhập tệp nhị phân

Kiểu	Ý nghĩa
“wb”	Mở tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã có nó sẽ bị xóa.
“rb”	Mở tệp mới để đọc theo kiểu nhị phân. Nếu tệp không có sẽ sinh ra lỗi.
“ab”	Mở tệp theo kiểu nhị phân để ghi bổ sung vào cuối tệp. Nếu tệp chưa có sẽ tạo tệp mới.
“r+b”	Mở tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp không có sẽ sinh ra lỗi.
“w+b”	Mở tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã có nó sẽ bị xóa.
“a+b”	Mở tệp theo kiểu nhị phân để đọc/ghi bổ sung vào cuối tệp. Nếu tệp chưa có sẽ tạo tệp mới.

1. Ghi dữ liệu ra tệp nhị phân (*tiếp*)

- B2: Ghi dữ liệu ra tệp bằng hàm fwrite()

  fwrite(ptr, size, n, fp);

  trong đó: +) ptr là con trỏ trỏ tới vùng nhớ chứa các phần tử dữ liệu cần ghi.

  +) size là kích thước phần tử theo byte.

  +) n là số phần tử cần ghi.

  +) fp là con trỏ tệp.

  Nếu có lỗi không ghi được, hàm trả về 0. Nếu không có lỗi hàm trả về số phần tử ghi được.

Ví dụ:

```
FILE *f = fopen(“songuyen.dat”,”wb”);
int a=200;
fwrite(&a,sizeof(a),1,f);
```


1. Ghi dữ liệu ra tệp nhị phân (*tiếp*)

- B3: Đóng tệp

  fclose(fp);

  trong đó fp là con trỏ tệp.

Ví dụ:

```
fclose(fp);
```


2. Đọc dữ liệu từ tệp nhị phân

- B1: Mở tệp để đọc bằng hàm fopen()

  fp = fopen(Tên tệp, Kiểu truy nhập);

  trong đó: +) fp là con trỏ tệp được khai báo trỏ tới kiểu FILE;

  +) Tên tệp có thể là hằng xâu hoặc biến xâu. Trong tên tệp có thể có đường dẫn.

  + Kiểu truy nhập tệp là hằng xâu diễn tả cách truy nhập vào tệp.

2. Đọc dữ liệu từ tệp nhị phân (*tiếp*)

- B2: Đọc dữ liệu từ tệp bằng hàm fread()

  fread(ptr, size, n, fp);

  trong đó: +) ptr là con trỏ trỏ tới vùng nhớ chứa các phần tử dữ liệu đọc được.

  +) size là kích thước phần tử theo byte.

  +) n là số phần tử cần đọc.

  +) fp là con trỏ tệp.

  Hàm fread đọc n phần tử của tệp kể từ vị trí con trỏ chỉ vị. Hàm trả về số phần tử đọc được.

2. Đọc dữ liệu từ tệp nhị phân (*tiếp*)

- B2: Đọc dữ liệu từ tệp bằng hàm fread()

  fread(ptr, size, n, fp);

  Nếu con trỏ chỉ vị đã ở cuối tệp (EOF) mà vẫn đọc sẽ sinh lỗi.

  Trước khi đọc tệp cần kiểm tra con trỏ chỉ vị đã ở cuối tệp chưa => dùng hàm feof(con trỏ tệp)

  Nên đọc từng phần tử tệp, trước khi đọc cần kiểm tra vị trí con trỏ chỉ vị.

2. Đọc dữ liệu từ tệp nhị phân (*tiếp*)

- B3: Đóng tệp

  fclose(fp);

  trong đó fp là con trỏ tệp.

Ví dụ:

```
fclose(fp);
```


Ví dụ

Viết chương trình tạo tệp “sonuyen.dat” chứa n số nguyên nhập vào từ bàn phím. Đọc lại các số nguyên từ tệp và đưa ra màn hình.

III. Tệp văn bản

1. Ghi dữ liệu ra tệp văn bản
2. Đọc lại dữ liệu từ tệp văn bản

1. Ghi dữ liệu ra tệp văn bản

B1: Mở tệp để ghi bằng hàm fopen()

fp = fopen(“Tên tệp”,”Kiểu truy nhập”);

trong đó: +) fp là con trỏ tệp được khai báo trỏ tới kiểu FILE;

+) Tên tệp có thể là hằng xâu hoặc biến xâu. Trong tên tệp có thể có đường dẫn.

+ Kiểu truy nhập tệp là hằng xâu diễn tả cách truy nhập vào tệp.

Các kiểu truy nhập tệp văn bản

Kiểu	Ý nghĩa
“wt”	Mở tệp mới để ghi theo kiểu văn bản. Nếu tệp đã có nó sẽ bị xóa.
“rt”	Mở tệp mới để đọc theo kiểu văn bản. Nếu tệp không có sẽ sinh ra lỗi.
“at”	Mở tệp theo kiểu văn bản để ghi bổ sung vào cuối tệp. Nếu tệp chưa có sẽ tạo tệp mới.
“r+t”	Mở tệp mới để đọc/ghi theo kiểu văn bản. Nếu tệp không có sẽ sinh ra lỗi.
“w+t”	Mở tệp mới để đọc/ghi theo kiểu văn bản. Nếu tệp đã có nó sẽ bị xóa.

1. Ghi dữ liệu ra tệp văn bản (*tiếp*)

B2: Ghi dữ liệu ra tệp văn bản

- Ghi dữ liệu theo định dạng ra tệp văn bản giống như đưa dữ liệu ra màn hình.

fprintf(fp,dk,…);

trong đó: +) fp là con trỏ tệp

+) dk là hằng xâu ký tự có chứa đặc tả chuyển dạng dữ liệu.

+) … là các đối mà giá trị của chúng cần ghi tệp.

Ví dụ: fprintf(fp,“x= %d y= %d”,x,y);

1. Ghi dữ liệu ra tệp văn bản (*tiếp*)

B2: Ghi dữ liệu ra tệp văn bản

- Ghi cả xâu ký tự ra tệp văn bản.

fputs(s,fp);

trong đó: +) fp là con trỏ tệp

+) s là hằng xâu hoặc biến xâu.

Hàm fputs() không ghi ký tự '\0' ra tệp.

Ví dụ: fputs("Hung Yen",fp);

1. Ghi dữ liệu ra tệp văn bản (*tiếp*)

B3: Đóng tệp

fclose(fp);

trong đó fp là con trỏ tệp.

2. Đọc lại dữ liệu từ tệp văn bản

B1: Mở tệp để đọc bằng hàm fopen()

fp = fopen("Tên tệp","Kiểu truy nhập");

trong đó: +) fp là con trỏ tệp được khai báo trỏ tới kiểu FILE;

+) Tên tệp có thể là hằng xâu hoặc biến xâu. Trong tên tệp có thể có đường dẫn.

+) Kiểu truy nhập tệp là hằng xâu diễn tả cách truy nhập vào tệp.

2. Đọc lại dữ liệu từ tệp văn bản (*tiếp*)

B2: Đọc lại dữ liệu từ tệp

- Đọc dữ liệu có định dạng

fscanf(fp, dk, …)

trong đó: +) fp là con trỏ tệp

+) dk là hằng xâu ký tự có chứa đặc tả chuyển dạng dữ liệu.

+) … là các địa chỉ vùng nhớ chứa dữ liệu đọc được.

Ví dụ: fscanf(fp,"%d",&b);

2. Đọc lại dữ liệu từ tệp văn bản (*tiếp*)

B2: Đọc lại dữ liệu từ tệp

- Đọc một xâu ký tự từ tệp

fgets(s, n, fp);

trong đó: +) s là biến xâu ký tự.

+) n là số ký tự tối đa sẽ đọc.

+) fp là con trỏ tệp

Ví dụ: fgets(s,20,fp);

2. Đọc lại dữ liệu từ tệp văn bản (*tiếp*)

B3: Đóng tệp

fclose(fp);

trong đó fp là con trỏ tệp.

IV. Truy nhập trực tiếp các phần tử của tệp

1. Các hàm di chuyển con trỏ chỉ vị

- Hàm rewind(fp) chuyển con trỏ về phần tử đầu tiên của tệp, fp là con trỏ tệp.
- Hàm fseek(fp, sb, xp)

trong đó: +) fp là con trỏ tệp

+) sb là số byte cần di chuyển

+) xp là vị trí xuất phát. xp chỉ có thể nhận một trong 3 giá trị sau:

SEEK_SET hoặc 0: xuất phát từ đầu tệp

SEEK_CUR hoặc 1: xuất phát từ vị trí hiện tại của con trỏ chỉ vị

SEEK_END hoặc 2: Xuất phát từ cuối tệp.

1. Các hàm di chuyển con trỏ chỉ vị (*tiếp*)

Hàm fseek() di chuyển con trỏ chỉ vị của tệp fp từ vị trí xác định bởi xp qua số byte sb. Chiều di chuyển về cuối tệp nếu sb dương, về đầu tệp nếu sb âm.

Hàm này trả về 0 nếu di chuyển thành công, trả về giá trị khác không nếu di chuyển không thành công.

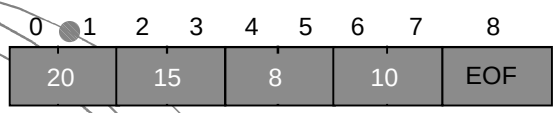
Chú ý: Không nên dùng fseek cho tệp văn bản.

1. Các hàm di chuyển con trỏ chỉ vị (*tiếp*)

- Hàm ftell(fp) cho biết vị trí hiện tại của con trỏ chỉ vị.

Ứng dụng: 1) Xác định kích thước tệp

2) Xác định số phần tử tệp



2. Truy nhập một phần tử bất kỳ của tệp

B1: Mở tệp với kiểu truy nhập là “r+b”

B2: Di chuyển con trỏ chỉ vị tới phần tử cần đọc/ghi

B3: Đọc/ghi phần tử

B4: Đóng tệp

3. Một số hàm thao tác trên tệp

- Hàm fcloseall() đóng tất cả các tệp đang mở.
- Hàm fflush(con trỏ tệp) làm sạch vùng đệm tệp.
- Hàm fflushall() làm sạch vùng đệm của tất cả các tệp đang mở.
- Hàm ferror(con trỏ tệp) kiểm tra lỗi thao tác tệp, nếu không có lỗi trả về 0, có lỗi trả về giá trị khác 0.
- Hàm remove(Tên tệp) xóa tệp