NGÔN NGỮ C CHUẨN (ANSI C)

[ ]--bao mục tùy chọn; fn--hàm; b--khối; rtn--trả về; ptd--trỏ đến; ptr--con trỏ; expr--biểu thức; TRUE--trị khác 0; FALSE--trị 0.

CÁC KIỂU DỮ LIỆU CƠ SỞ

char	Ký tự đơn (có dấu hoặc không dấu), 1byte
unsigned char	Ký tự không âm
short	Số nguyên ngắn
unsigned short	Số nguyên ngắn không âm
int	Số nguyên (giới hạn khai báo trong limits.h)
unsigned int	Số nguyên không âm
long	Số nguyên dài
unsigned long	Số nguyên dài không âm
float	Số thực (giới hạn khai báo trong float.h)
double	Số thực, độ chính xác kép
long double	Số thực, độ chính xác bội kép
void	Không kiểu; Dùng báo: 1) không trị trả về từ fn 2) không đối số cho fn 3) con trỏ cơ sở chung

CHUYỂN ĐỔI KIỂU DỮ LIỆU TRONG BIỂU THỨC

1. long double với toán hạng khác, toán hạng đó thành long double.
2. double với toán hạng khác, toán hạng đó thành double.
3. float với toán hạng khác, toán hạng đó thành float.
4. Tất cả char và short thành int nếu trình bày được dưới dạng int, nếu không thành unsigned int.
5. unsigned long với toán hạng khác, toán hạng đó thành unsigned long.
6. unsigned int với long và unsigned int trình bày được dưới dạng long, kiểu chung là long, nếu không là unsigned long.
7. long với toán hạng khác, toán hạng đó thành long.
8. unsigned int với toán hạng khác, toán hạng đó thành unsigned int.
9. Còn lại thành int.

CÁC PHÁT BIỂU

PHÁT BIỂU	MÔ TẢ
{ *local_var_decl* *statement* **... }**	Khối (block). Các khai báo biến cục bộ (*local_var_decl*) là tùy chọn.
break;	Chấm dứt thực hiện **for**, **while**, **do**, **switch**.
continue;	Bỏ qua các phát biểu theo sau **continue** trong **do**, **for**, **while**; và tiếp tục thực hiện vòng lặp.
do *statement* **while** (*expr*);	Thực hiện *statement* cho đến khi *expr* là FALSE; *statement* được thực hiện ít nhất một lần.
expr;	Định trị *expr*; bỏ qua kết quả.
for (*e1*; *e2*; *e3*) *statement* *e1:* khởi tạo *e2:* điều kiện *e3:* cập nhật	Định trị expr *e1* một lần; rồi lặp: định trị *e2*, thực hiện *statement*, và *e3* (theo thứ tự) cho đến khi *e2* là FALSE; chú ý *statement* sẽ không thực hiện nếu *e2* là FALSE ngay lần định trị đầu; *e1*, *e2* và *e3* là tùy chọn; khi thiếu *e2* thì *e2=1* được ngầm định.
goto *label*;	Phân nhánh phát biểu đến *label*, phải cùng trong hàm với **goto**. Vd: **int fn(void) {** **... goto write; ...** **write: printf("Here am I"); ...}**
if (*expr*) *statement*	Nếu *expr* là TRUE, thực hiện *statement*; nếu không thì bỏ qua *statement*;
if (*expr*) *statement1* **else** *statement2*	Nếu *expr* là TRUE, thực hiện *statement1*; nếu không thì thực hiện *statement2*;
;	Phát biểu rỗng. Không làm gì. Vd: **while (t[i++]);**
return *expr*;	Trả về từ hàm gọi trị của *exp*; *expr* thiếu trong trường hợp hàm **void**.
switch (*expr*) **{** **case** *const1*: *statement* **...; break**; **case** *const2*: *statement* **...; break;** **...** **default:** *statement* **... }**	*expr* (phải là một biểu thức nguyên) và được so sánh lần lượt với các expr hằng nguyên *const1*, *const2*, ... Nếu so trùng, thì phát biểu theo sau **case** (và trước **break** kế tiếp, nếu có) sẽ được thực hiện. Nếu không so trùng, thì phát biểu trong trường hợp **default** (nếu có) sẽ được thực hiện.
while (*expr*) *statement*	Thực hiện *statement* trong lúc *expr* còn TRUE; *statement* có thể không được thực hiện nếu *expr* là FALSE trong lần định trị đầu.
/ comment */*	Chú thích (một dòng hay trên nhiều dòng)

ĐỊNH NGHĨA KIỂU

typedef gán một tên mới cho một kiểu dữ liệu để tạo một tên dùng riêng cho kiểu dữ liệu đó. Vd:

```
/* gán kiểu struct vô danh với tên COMPLEX */
typedef struct {
  float real, imaginary;
} COMPLEX;
/* khai báo các biến với tên kiểu mới COMPLEX */
COMPLEX c1, c2, sum;
```

CÁC HẰNG SỐ

char	**'**		**'a'** **'\n'** (ký tự escape)
char string	**"**		**"hello"** **""** (chuỗi rỗng)
float	**...f,...F**	*(1)*	**7.2f** **2.e-15F** **-1E9f** **.5F**
double		*(1)*	**7.2** **2.e-15** **-1E9** **.5**
long double	**...l,...L**	*(1)*	**7.2l** **2.e-15l** **-1E9L** **.5L**
enumeration		*(2)*	**red** **january** **monday**
int			**17** **-5** **0**
long int	**...l,...L**	*(3)*	**251l** **100L**
unsigned int	**...u,...U**		**17u** **5U** **0u** **65535u**
hex integer	**0x,0X**		**0xFF** **0Xff** **0xA000l**
octal int	**0**		**0777** **0100U** **0573ul**

GHI CHÚ

(1) Ghi kiểu dấu chấm thập phân và (hoặc) kiểu khoa học.
(2) Các định danh khai báo trước cho kiểu liệt kê; trị xem như kiểu int.
(3) Hoặc bất kỳ kiểu int nào lớn hơn kiểu thông thường.

GIỚI HẠN KIỂU

const	Đối tượng hằng, không thể thay đổi bởi chương trình.
volatile	Phần cứng hay phần mềm bên ngoài có thể làm thay đổi biến này, không nên tối ưu hóa.

CÁC TOÁN TỬ

TOÁN TỬ	MÔ TẢ	VÍ DỤ	LIÊN KẾT
++	Tăng sau	**ptr++**	
- -	Giảm sau	**count- -**	
[ ]	Chỉ số mảng	**values [10]**	⇒
( )	Gọi hàm	**sqrt (x)**	
.	Truy cập struct member	**child.name**	
->	ptr đến struct member	**child_ptr->name**	
sizeof	Kích thước (bytes)	**sizeof child**	
++	Tăng trước	**++ptr**	
- -	Giảm trước	**- -count**	
&	Địa chỉ của	**&x**	
*****	Nội dung nơi ptr trỏ đến	***ptr**	⇐
+	Số dương	**+a**	
-	Số âm	**-a**	
~	Đảo bit (bù 1)	**~077**	
!	NOT logic	**! ready**	
(*type*)	Ép kiểu	**(float) total/n**	
*****	Nhân	**i * j**	
/	Chia	**i / j**	⇒
%	Phần dư (modulo)	**i % j**	
+	Cộng	**value + i**	⇒
-	Trừ	**x - 100**	
<<	Dịch trái	**byte << 4**	⇒
>>	Dịch phải	**i >> 2**	
<	Nhỏ hơn	**i < 100**	
<=	Nhỏ hơn hoặc bằng	**i <= j**	⇒
>	Lớn hơn	**i > 0**	
>=	Lớn hơn hoặc bằng	**count >= 90**	
==	Bằng	**result == 0**	⇒
!=	Không bằng (khác)	**c != EOF**	
&	AND bit	**word & 077**	⇒
^	XOR bit	**word1 ^ word2**	⇒
|	OR bit	**word | bits**	⇒
&&	AND logic	**j>0 && j<10**	➔
||	OR logic	**i>80 || ready**	➔
? :	Toán tử điều kiện	**a>b ? a : b** Nếu **a>b** thì expr=**a**, còn không expr=**b**.	⇐
= *= /= **%= += -=** **&= ^= |=** **<<= >>=**	Các toán tử gán	**count += 2** Tương đương: **count=count+2**	⇐
,	Toán tử dấu phẩy	**i=10 , j=0**	➔

GHI CHÚ

Các toán tử được liệt kê với thứ tự ưu tiên giảm dần.
Các toán tử cùng vùng có cùng thứ tự ưu tiên.
Thứ tự liên kết: ⇒ nhóm; ➔ tính toán theo đúng thứ tự này (Vd: **a = b = c;** sẽ được nhóm từ phải sang trái, tức: **a = (b = c);**).

CÁC PHÁT BIỂU TIỀN XỬ LÝ

PHÁT BIỂU	MÔ TẢ
#define *id text*	*text* sẽ được thay thế cho *id* tại nơi *id* xuất hiện trong chương trình (Vd: **#define BUFFERSIZE 512**). Nếu cấu trúc **id(a1,a2,...)** được dùng, các đối số **a1**,**a2**,... sẽ được thay thế nếu xuất hiện *id* có các đối số tương ứng khi gọi macro (Vd: **#define max(A,B) ((A)>(B)?(A):(B))**, nếu có **x=max(p+q,r+s)** macro sẽ thay thành **x=(p+q)>(r+s)?(p+q):(r+s)**)
#undef *id*	Loại bỏ định nghĩa (undefine) của *id*
#if *expr* **...** **#endif**	Nếu biểu thức hằng *expr* là TRUE, phát biểu trước **#endif** sẽ được xử lý, nếu không thì bỏ qua.
#if *expr* **...** **#else** **...** **#endif**	Nếu biểu thức hằng *expr* là TRUE, phát biểu trước **#endif** sẽ được xử lý, nếu không thì phát biểu giữa **#else** và **#endif** sẽ được xử lý.
#ifdef *id* **...** **#endif**	Nếu *id* được định nghĩa (với **#define** hay trên dòng lệnh), phát biểu trước **#endif** sẽ được xử lý, nếu không thì bỏ qua. (tùy chọn **#else** như trên)
#ifndef *id* **...** **#endif**	Nếu *id* không được định nghĩa, phát biểu trước **#endif** sẽ được xử lý, nếu không thì bỏ qua. (tùy chọn **#else** như trên)
#include "*file***"**	Chèn nội dung *file* vào chương trình; tìm trong thư mục chương trình, rồi tìm nơi lưu trữ chuẩn.
#include <*file***>**	Chèn nội dung *file* vào chương trình; chỉ tìm trong nơi lưu trữ chuẩn.
#line *n* **"***file***"**	Chỉ định những dòng tiếp theo của chương trình là lấy từ *file*, kể từ dòng thứ *n*; *file* tùy chọn.

GHI CHÚ

Các phát biểu tiền xử lý có thể trải trên nhiều dòng, dòng chưa kết thúc sẽ có dấu (\) ở cuối. Các phát biểu cũng có thể lồng nhau.

CÁC LỚP LƯU TRỮ

LỚP LƯU TRỮ	KHAI BÁO	CÓ THỂ ĐƯỢC THAM CHIẾU	KHỞI TẠO VỚI	GHI CHÚ
static	ngoài fn	mọi nơi trong file	biểu thức hằng	1
	trong fn/b	trong fn/b	biểu thức hằng	1
extern	ngoài fn	mọi nơi trong file	biểu thức hằng	2
	trong fn/b	trong fn/b	không khởi tạo	2
auto	trong fn/b	trong fn/b	biểu thức bất kỳ	3
register	trong fn/b	trong fn/b	biểu thức bất kỳ	3, 4, 6
(không có)	ngoài fn	mọi nơi trong file, file khác với extern	biểu thức hằng	5
	trong fn/b	trong fn/b	biểu thức bất kỳ	3, 6

GHI CHÚ

1. Khởi tạo ngay khi chương trình thực hiện; mặc định là 0.
2. Biến phải được định nghĩa chỉ một nơi với **extern**.
3. Biến khởi tạo mỗi lần fn/b được gọi; không có trị mặc định.
4. Việc gán cho thanh ghi không bảo đảm; hạn chế kiểu (phụ thuộc thi công) có thể được gán đến thanh ghi. Không áp dụng được toán tử **&**.
5. Biến chỉ khai báo chỉ một nơi. Khởi tạo ngay khi chương trình thực hiện; mặc định là 0.
6. Mặc định là **auto**.

CÁC BIỂU THỨC

Một biểu thức gồm một hay nhiều mục dữ liệu và có hoặc không có toán tử. Một mục dữ liệu có thể là:
- *name* (tên hàm hoặc tên đối tượng dữ liệu)
- *constant*
- **sizeof**(*type*)
- (*expr*)

Một biểu thức là biểu thức hằng nếu các mục dữ liệu của nó là hằng.

CÁC MẢNG

Mảng một chiều **aname** với **n** phần tử có kiểu ***type*** và được khởi tạo với các trị (tùy chọn) sẽ được khai báo như sau:

```
type aname[n] = { val1, val2, ... };
```

Nếu chỉ định đầy đủ các trị, **n** có thể bỏ qua.
Chỉ có mảng static hay toàn cục mới tự khởi tạo.
Mảng char có thể được khởi tạo bởi một chuỗi ký tự trong ngoặc kép **“”**.
Chỉ số có nghĩa của mảng là từ **0** đến **n-1**.
Mảng nhiều chiều được khai báo như sau:

```
type aname[n1][n2]... = { init_list };
```

Các trị liệt kê trong danh sách khởi tạo được gán theo “thứ tự chiều” (nghĩa là chiều cuối cùng sẽ tăng trước). Các cặp { } lồng có thể dùng thay thế thứ tự này nếu cần.
VÍ DỤ

```
/* mảng char */
static char hisname[] = {"John Smith"};
/* mảng các ptr đến char */
static char *days[7] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
/* mảng 3x2 các số int */
int matrix[3][2] = { {10,11},{-5,0}, {11,21} };
/* mảng struct complex */
struct complex sensor_data[100];
```

POINTER

Một biến có thể khai báo thành một pointer trỏ đến kiểu chỉ địnhbằng phát biểu có dạng sau:

```
type *name;
```

VÍ DỤ

```
/* numptr trỏ đến số float */
float *numptr;
/* pointer trỏ đến struct complex */
struct complex *cp;
/* nếu phần thực của struct complex được trỏ bởi cp bằng 0.0 */
if (cp->real == 0.0) {...}
/* sptr trỏ đến char; đặt bằng địa chỉ của buf[25] (tức trỏ đến buf[25]) */
char *sptr = &buf[25];
*sptr = 'c';                /* lưu 'c' vào vị trí trỏ bởi sptr */
++sptr;                     /* cho sptr trỏ đến vị trí kế tiếp trong buf */
int (*fptr) ();             /* pointer trỏ đến hàm trả về kiểu int */
```

CÁC HÀM

Các hàm có dạng như sau:

```
ret_type name (arg1_decl, arg2_decl, ... )
{
   local_var_decl
   statement
   ...
   return value;
}
```

Các hàm có thể được khai báo **extern** (mặc định) hoặc **static**.
Các hàm **static** chỉ có thể được gọi trong file nó được đinh nghĩa.
ret_type kiểu trị trả về của hàm, có thể là **void** nếu không trả về trị nào.
VÍ DỤ

```
/* fn để tìm chiều dài một chuỗi ký tự */
int strlen (char *s)
{
   int length = 0;
   while ( *s++ ) ++length;
   return length;
}
```

CÁC STRUCTURE

Các biến thành viên của một structure **sname** được khai báo dưới dạng:

```
struct sname
{
   member_declaration;
   ...
} variable_list;
```

Mỗi thành viên được khai báo gồm kiểu theo sau là tên một hay nhiều thành viên;
Một trường **mname** rộng **n**-bit được khai báo với phát biểu có dạng sau:

```
type mname:n;
```

Nếu thiếu **mname**, **n** bit không được đặt tên sẽ được dành sẵn; nếu **n** cũng bằng 0, trường tiếp sẽ được đặt trọn vào một word.
variable_list (tùy chọn) dùng khai báo các biến của kiểu structure này.
Nếu **sname** đã được cung cấp, các biến cũng có thể khai báo sau bằng cách dùng dạng:

```
struct sname variable_list;
```

VÍ DỤ

```
/* khai báo structure complex */
struct complex {
  float real, imaginary;
};
/* định nghĩa các structure */
struct complex c1 = { 5.0 , 0.0 };
struct complex c2, csum;
c2 = c1;                    /* gán c1 cho c2 */
csum.real = c1.real + c2.real;
```

CÁC UNION

Các biến thành viên của một union **uname** chiếm cùng một vùng nhớ và được khai báo dưới dạng:

```
union uname
{
   member_declaration;
   ...
} variable_list;
```

Mỗi thành viên được khai báo gồm kiểu theo sau là tên một hay nhiều thành viên;
variable_list (tùy chọn) dùng khai báo các biến của kiểu union này. Nếu **uname** đã được cung cấp, các biến cũng có thể khai báo sau bằng cách dùng dạng: **union uname variable_list;**
GHI CHÚ: Các union không được khởi tạo trước.

CÁC KIỂU DỮ LIỆU LIỆT KÊ (ENUM)

Kiểu dữ liệu liệt kê **ename** với các trị **enum1,enum2**,… được khai báo:
enum ename { enum1, enum2, ... } variable_list;
variable_list (tùy chọn) dùng khai báo các biến của kiểu enum này.
Từng trị được liệt kê là một định danh theo sau (tùy chọn) bởi dấu bằng và một biểu thức hằng. Các trị tuần tự bắt đầu từ **0** được gán cho các trị trên bởi trình biên dịch, nếu không cấu trúc **enum=value** sẽ được dùng.
Nếu có **ename**, các biến cũng có thể khai báo sau bằng cách dùng dạng:
enum ename variable_list;
VÍ DỤ
enum boolean { false, true }; /* định nghĩa kiểu boolean */
enum boolean done = false; /* khai báo biến và trị khởi tạo */
if (done==true) {...} /* kiểm tra trị */

ĐỊNH DẠNG DỮ LIỆU XUẤT

printf dùng để ghi dữ liệu ra thiết bị xuất chuẩn (thường là màn hình). Để ghi ra file, dùng **fprintf**; để "ghi" dữ liệu vào mảng ký tự, dùng **sprintf**.
Dạng tổng quát của **printf** là:
printf (format, arg1, arg2, ...)
ở đây **format** là chuỗi ký tự định dạng mô tả cách **arg1,arg2**,… được in.
Dạng tổng quát của một mục trong chuỗi định dạng này là:
%[flags][size][.prec]type
flags:
- canh trái trị (mặc định là canh phải)
+ đặt một dấu **+** hoặc **-** trước trị
space đặt một dấu cách trước trị dương
đặt **0** trước trị bát phân, **0X** trước trị thập lục phân; đặt dấu chấm thập phân cho trị float, và bỏ qua những số 0 dư cho kiểu **g** hoặc **G**
0 hiển thị những số 0 ở đầu
size: số chỉ định kích thước tối thiểu của trường; có thể cung cấp từ đối số (kiểu int) và dùng * để giữ chỗ cho đối số đó trong chuỗi định dạng
prec: số tối thiểu các con số sẽ xuất hiện cho mỗi số **int**; số các số thập phân cho **e** và **f**; số tối đa các số có nghĩa cho **g**; số tối đa các ký tự cho **s**; nếu * thì sẽ lấy số chỉ định từ đối số (int) tương ứng
type: chỉ định kiểu trị được hiển thị theo mã ký tự sau:

arg	dec.	oct.	hex.	HEX.	±d.dd	±d.dde±dd
short	**hd**					
unsigned short	**hu**	**ho**	**hx**	**hX**	*mặc định độ chính*	
int	**d**				*xác 6 số thập phân*	
unsigned int	**u**	**o**	**x**	**X**		
long	**ld**					
unsigned long	**lu**	**lo**	**lx**	**lX**		
float, double					**f**	**e**
long double					**Lf**	**Le**

i giống như **d**
p một con trỏ, void * (định nghĩa khi thực thi)
n lưu số ký tự có thể được hiển thị, đối số là int *, không xuất
hn lưu số ký tự có thể được hiển thị, đối số là short *, không xuất
ln lưu số ký tự có thể được hiển thị, đối số là long *, không xuất
E giống như **e** nhưng hiển thị **E** trước số mũ thay cho **e**
g một double trong dạng **f** hoặc **e**, với cái nào ngắn hơn
G một double trong dạng **f** hoặc **E**, với cái nào ngắn hơn
c một ký tự
s một chuỗi ký tự kết thúc bằng null
% in %
GHI CHÚ
các ký tự trong chuỗi định dạng không có % sẽ được in đúng như vậy;
dạng thức dấu chấm động hiển thị cho cả float và double;
dạng thức số nguyên có thể hiển thị char, short int hoặc int.
VÍ DỤ
printf("%o + %#X = %+0*d", 31, 31, 5, 31+31);
Kết quả: **37 + 0X1F = +0062**
printf("%f %g %#.0f %.2g", 3.14, 3.14, 3.14, 3.14);
Kết quả: **3.140000 3.14 3. 3.1**

ĐỊNH DẠNG DỮ LIỆU NHẬP

scanf dùng để đọc dữ liệu từ ngõ nhập chuẩn. Để đọc dữ liệu từ một file, dùng **fscanf**. Để "đọc" dữ liệu từ một mảng ký tự, dùng **sscanf**. Dạng tổng quát của **scanf** là:
scanf (format, arg1, arg2, ...)
ở đây **format** là chuỗi ký tự định dạng mô tả dữ liệu được đọc, **arg1, arg2, ...** chỉ đến nơi dữ liệu đọc vào được lưu trữ. Dạng thức của một mục trong chuỗi định dạng này là:
%[*][size]type
***:** đặt trước mã định dạng, sẽ đọc vào một vùng nhưng không gán nó
size: số cho biết kích thước tối đa của trường nhập
type: chỉ định kiểu của trị được đọc:

arg là ptr đến	dec.	oct.	hex.	HEX.	±d.dd	±d.dde±dd
short	**hd**					
unsigned short	**hu**	**ho**	**hx**	**hX**		
int	**d**					
unsigned int	**u**	**o**	**x**	**X**		
long	**ld**					
unsigned long	**lu**	**lo**	**lx**	**lX**		
float					**f,e,E,g,G**	
double					**lf,le,lE,lg,lG**	
long double					**Lf,Le,LE,Lg,LG**	

i giống như **d**
p một con trỏ (như trong **printf**), đối số kiểu void **
n lưu số ký tự đã so trùng, đối số là int *, không nhập
hn lưu số ký tự đã so trùng, đối số là short *, không nhập
ln lưu số ký tự đã so trùng, đối là long *, không nhập
c một ký tự, đối số là char[ ]
s một chuỗi ký tự kết thúc bằng dấu cách, đối số là char[ ]
% %
[...] chuỗi ký tự, kết thúc bởi ký tự bất kỳ không liệt kê trong [**v**à **]**; nếu ký tự đầu trong **[]** là **^**, chấp nhận các ký tự không liệt kê trong **[ ]**
GHI CHÚ
Một hàm scan trả về khi:
- Nó tìm thấy ký tự NULL kết thúc trong chuỗi định dạng
- Ký tự nhập không phải kiểu chờ đợi
- Chuyển kiểu thất bại
Bất kỳ ký tự nào trong chuỗi định dạng không bắt đầu bằng % sẽ được chờ so trùng tại đầu nhập. Vd: **scanf("value=%d",&ival);** so trùng chuỗi **"value="** tại đầu nhập, tiếp theo đọc và lưu một số nguyên trong **ival**.
Một ký tự trắng trong chuỗi định dạng làm cho hàm bỏ qua một hoặc nhiều ký tự trắng trong dòng nhập.
VÍ DỤ
sscanf("12Free of charge 21", "%X%c%*[^ab]%2s%d",&i,&c,text,&j);
sẽ trả về **3** và **i=303**, **c='r'**, **text="ar"**; **j** vẫn không thay đổi.

CÁC KÝ TỰ ESCAPE

\b	Dấu cách (*BS*)	****	Backslash (\)
\f	Máy in sang trang mới (*FF*)	*****nnn*	Trị ký tự hệ bát phân (*n*: **0-7**)
\n	Tạo dòng mới (*NL*)	**\x***hh*	Trị ký tự hệ thập lục phân
\r	Xuống dòng (*CR*)		(*h*: **0-9, a-f, A-F**)
\t	Tab ngang (*HT*)	**\"**	Nháy kép (")
\v	Tab dọc (*VT*)	**\'**	Nháy đơn (')
\a	Tiếng chuông (*BEL*)	**\?**	Dấu hỏi (?)

THƯ VIỆN CÁC HÀM VÀ MACRO

Các kiểu đối số của hàm:

int c; /* **char** */	**int n,n1,n2;**
unsigned int u;	**long l,l1,l2;**
double d,d1,d2;	**char *s,*s1,*s2;**
FILE *f;	**size_t su,su1,su2;**
time_t tl,tl1,tl2;	**fpos_t fl;**
void *v,*v1,*v2;	**va_list ap;**

char và **short** được chuyển thành **int** khi truyền đến hàm;
float được chuyển thành **double**.
/…/ trả về mã lỗi (…) trả về mã thành công

Phân loại ký tự — ctype.h

int isalnum(c)	TRUE nếu **c** là ký tự chữ số (alphanumeric) bất kỳ
int isalpha(c)	TRUE nếu **c** là ký tự chữ cái (alphabetic) bất kỳ
int iscntrl(c)	TRUE nếu **c** là ký tự điều khiển bất kỳ
int isdigit(c)	TRUE nếu **c** là số thập phân bất kỳ từ **0-9**
int isgraph(c)	TRUE nếu **c** là ký tự in được bất kỳ trừ *space*
int islower(c)	TRUE nếu **c** là ký tự in thường bất kỳ
int isprint(c)	TRUE nếu **c** là ký tự in được bất kỳ kể cả *space*
int ispunct(c)	TRUE nếu **c** không là ký tự điều khiển lẫn ký tự số
int isspace(c)	TRUE nếu **c** là một trong các ký tự khoảng trắng: space, *FF, NL, CR, HT, VT*
int isupper(c)	TRUE nếu **c** là ký tự in hoa bất kỳ
int isxdigit(c)	TRUE nếu **c** là số hex bất kỳ **0-9, A-F, a-f**
int tolower(c)	chuyển **c** thành chữ thường
int toupper(c)	chuyển **c** thành chữ hoa

Chuyển đổi data — stdlib.h

double atof(s)	chuyển chuỗi ASCII thành double /HUGE_VAL,0/
int atoi(s)	chuyển chuỗi ASCII thành int
long atol(s)	chuyển chuỗi ASCII thành long
double strtod(s1,*s2)	chuyển chuỗi ASCII thành double; khi trả về, ***s2** trỏ ký tự đầu tiên sau chuỗi đổi trong **s1**
long strtol(s1,*s2,n)	chuyển chuỗi ASCII thành long, cơ số **n**; khi trả về, ***s2** trỏ ký tự đầu tiên sau chuỗi đổi trong **s1**
unsigned long strtoul(s1,*s2,n)	chuyển chuỗi ASCII thành unsigned long (xem **strol**)

Xử lý file và xuất/nhập — stdio.h

void clearerr(f)	đặt lại lỗi (kể cả EOF) trên file
int fclose(f)	đóng file /EOF/ (0)
int feof(f)	TRUE nếu end-of-file trên **f**
int ferror(f)	TRUE nếu có lỗi I/O trên **f**
int fflush(f)	ghi ngay vùng đệm xuất đến **f** /EOF/ (0)
int fgetc(f)	đọc ký tự kế tiếp từ **f** /EOF/
int fgetpos(f,*fl)	lấy vị trí trong file chỉ bởi **fl** /TRUE/ (0)
char *fgets(s,n,f)	đọc **n-1** ký tự từ **f**, trừ khi thấy newline hoặc end-of-file; newline được lưu trữ trong **s** nếu đọc /NULL/
FILE *fopen(s1,s2)	mở file **s1**, chế độ **s2**: "w"=ghi, "r"=đọc, "a"=nối sau "b"=nhị phân, "+"=cập nhật /NULL/
int fprintf(f,s,...)	ghi các đối số đến **f** dùng định dạng **s** (xem **printf**)
int fputc(c,f)	ghi **c** đến **f**; rtn **c** /EOF/
int fputs(s,f)	ghi **s** đến **f** /EOF/ (≥0)
size_t fread (v,su1,su2,f)	đọc **su2** mục dữ liệu từ **f** vào **v**; **su1** là số byte của mỗi mục dữ liệu /0/ (số byte đọc/**su1**)
FILE *freopen(s1,s2,f)	đóng file **f** và mở **s1** với chế độ **s2** (xem **fopen**)
int fscanf(f,s,...)	đọc các đối số từ **f** dùng định dạng **s** (xem **scanf**)
int fseek(f,l,n)	vị trí con trỏ file, nếu **n=SEEK_SET**, **l** là khoảng cách kể từ đầu; nếu **n=SEEK_CUR**, kể từ vị trí hiện tại, nếu **n=SEEK_END**, kể từ cuối file /TRUE/ (0)
int fsetpos(f,*fl)	đặt vị trí file vào **fl** (0) /TRUE/
long ftell(f)	khoảng cách hiện tại kể từ đầu file /-1L/
size_t fwrite (v,su1, su2, f)	ghi **su2** mục dữ liệu từ **v** vào **f**; **su1** là số byte của mỗi mục dữ liệu /0/ (số byte ghi/**su1**)
int getc(f)	đọc ký tự kế tiếp từ **f** /EOF/
int getchar()	đọc ký tự kế tiếp từ **stdin** /EOF/
char *gets(s)	đọc chuỗi ký tự từ **stdin** vào **s** đến khi thấy newline hoặc end-of-file; không lưu trữ newline /NULL/
void perror(s)	xuất **s** theo sau là mô tả lỗi cuối, đến **stderr**
int printf(s,...)	ghi các đối số đến **stdout** với định dạng **s**; trả về số ký tự được ghi /<0/
int putc(c,f)	ghi **c** đến **f**; rtn **c** /EOF/
int putchar(c)	gọi **fputc(c,stdout)**
int puts(s)	ghi **s** và newline đến **stdout** /EOF/ (≥0)
int remove(s)	xóa file có tên **s** (0) /TRUE/
int rename(s1,s2)	đổi tên file có tên **s1** thành file có tên **s2** (0) /-1/
void rewind(f)	quay lại đầu file **f**; gọi **fseek(f,0L,SEEK_SET)**
int scanf(s,...)	đọc các đối số từ **stdin** với định dạng **s**; trả về số trị đọc được hoặc EOF
void setbuf(f,s)	nếu **s<>NULL** gọi **setvbuf(f,s,_IOFBF,BUFSIZ)**, nếu không gọi **setvbuf(f,NULL,_IONBF,BUFSIZ)**
int setvbuf(f,s,n,su)	đặt chế độ đệm cho **f**, vùng đệm là **s** kích thước **su**, **n** là một trong **_IOFBF** (full buffering), **_IOLBF** (line buffering), **_IONBF** (no buffering) (0) /TRUE/
int sprintf(s1,s2,...)	ghi các đối số đến vùng đệm **s1**, với định dạng **s2**
int sscanf(s1,s2,...)	đọc các đối số từ **s1**, với định dạng **s2** (xem **scanf**)
FILE *tmpfile()	tạo file tạm, chế độ mở "wb+", rtn ptr đến nó /NULL/
char *tmpnam(s)	sinh tên file tạm; đặt kết quả trong **s** nếu **s<>NULL** (kích thước vùng đệm **L_tmpnam**); rtn ptr đến tên
int ungetc(c,f)	chèn **c** trở lại file **f** (nếu **c** không đọc được) /EOF/
int vfprintf(f,s,ap)	xem **vprintf** và **fprintf**
int vprintf(s,ap)	giống **printf** với danh sách biến đối số **ap**; **va_start** phải được gọi trước và **va_end** sau khi gọi hàm
int vsprintf(s1,s2,ap)	xem **vprintf** và **sprintf**

Toán học — math.h,stdlib.h(*)

int errno (errno.h)		xác định phạm vi lỗi (ERANGE) và vùng lỗi (EDOM)
int abs(n)	*	trị tuyệt đối của **n**
double acos(d)		arccosin của **d** /0/ [0,π]
double asin(d)		arcsin của **d** /0/ [-π/2,+π/2]
double atan(d)		arctang của **d** [-π/2,+π/2]
double atan2(d1,d2)		arctang của **d1/d2** [-π,+π]
double ceil(d)		số nguyên nhỏ nhất không nhỏ hơn **d**
double cos(d)		cosin của **d** (**d** bằng radians) (rad = deg* π/180)
double cosh(d)		hiperbolic cosin của **d**
div_t div(n1,n2)	*	tính thương số và số dư của phép chia **n1/n2**
double exp(d)		**e** lũy thừa **d** /HUGE_VAL/
double fabs(d)		trị tuyệt đối của **d**
double floor(d)		số nguyên lớn nhất không lớn hơn **d**
double fmod(d1,d2)		số dư phép chia **d1/d2**
double frexp(d,*n)		trả về **x** trong nửa đoạn [½,1), và **d=x*2^n**
long labs(l)	*	trị tuyệt đối của **l**
double ldexp(d,n)		**d*2^n**
ldiv_t ldiv(l1,l2)	*	tính thương số và số dư của phép chia **l1/l2**
double log(d)		log tự nhiên của **d** /0/
double log10/(d)		log cơ số 10 của **d** /0/
double modf(d1,*d2)		rtn **x** với **d1=x+d2**, **x** trong [0,1), **d2** nguyên
double pow(d1,d2)		**d1** lũy thừa **d2** /0,HUGE_VAL/
int rand()	*	số ngẫu nhiên trong đoạn [0,RAND_MAX]
double sin(s)		sin của **d** (**d** bằng radians)
double sinh(d)		hyperbolic sin của **d**
double sqrt(d)		căn bậc hai của **d** /0/
void srand(u)	*	đặt lại bộ sinh số ngẫu nhiên với **u**
double tan(d)		tang của **d** (radians) /HUGE_VAL/
double tanh(d)		hyperbolic tang của **d**

Thao tác và cấp phát vùng nhớ — string.h,stdlib.h(*)

void *calloc (su1,su2)	*	cấp phát vùng nhớ cho **su1** phần tử; mỗi phần tử chiếm **su2** byte và khởi tạo bằng 0 /NULL/
void free(v)	*	giải phóng vùng nhớ do **v** trỏ đến
void *malloc(su)	*	cấp phát **su** byte và trả về ptr đến nó /NULL/
void *memchr (v,c,su)		rtn ptr trong **v** đến ký tự đầu tiên là **c**, dò tìm tối đa **su** unsigned char, hoặc NULL nếu không tìm thấy
int memcmp (v1,v2,su)		rtn **<0**, **=0**, **>0** nếu v1 **<**,**=** hoặc **>v2**, so sánh tối đa **su** ký tự
void *memcpy (v1,v2,su)		sao chép **su** ký tự từ **v2** đến **v1** (**v1**, **v2** phải không chồng lắp); trả về **v1**
void *memmove (v1,v2,su)		sao chép **su** ký tự từ **v2** đến **v1** (**v1**, **v2** có thể chồng lắp); trả về **v1**
void memset(v,c,su)		đặt **c** vào **su** ký tự đầu của **v**; rtn **v**
void *realloc(v,su)	*	đổi kích thước **v** thành **su** và rtn ptr đến nó /NULL/

Điều khiển chương trình — setjmp.h,stdlib.h(*)

void assert(*iexpr*) (assert.h)		nếu NDBUG không định nghĩa và **iexpr** là FALSE ghi thông tin chẩn đoán đến **stderr** và gọi **abort()**
void abort()	*	làm chương trình kết thúc không bình thường
int atexit (void(*func)(void))	*	đăng ký **func**, sẽ được gọi khi thoát (0) /TRUE/
void exit(n)	*	chấm dứt thực thi, trả về trạng thái thoát **n**
char *getenv(s)	*	rtn ptr đến trị của tên môi trường **s** /NULL/
void longjmp (jmp_buf env,n)		phục hồi môi trường từ **env**; **setjmp** trả về **n** nếu có hoặc **1** nếu **n=0**
int setjmp (jmp_buf env)		lưu stack môi trường trong **env**; (0) (xem **longjmp**)
int system(s)	*	chạy **s** như nhập từ terminal; rtn trạng thái thoát /-1/

Tìm kiếm và Sắp xếp — stdlib.h

void *bsearch (void *key, void *base, su1, su2, int (*cmp) (void *ck, void *ce))	tìm kiếm nhị phân trong mảng **base** (**su1** phần tử, mỗi phần tử chiếm **su2** bytes) dùng hàm **cmp** để so sánh; **cmp** phải trả về trị âm nếu **ck<ce**, **0** nếu **ck=ce**, dương nếu **ck>ce**
void qsort(void *base, su1, su2, int (*cmp) (void *ck, void *ce))	sắp xếp kiểu quicksort mảng **base** (**su1** phần tử, mỗi phần tử chiếm **su2** bytes), dùng hàm **cmp** để so sánh; (xem **bsearch**)

Thao tác chuỗi — string.h

char *strcat(s1,s2)	nối **s2** vào cuối **s1**; rtn **s1**
char *strchr(s,c)	rtn ptr đến ký tự **c** xuất hiện đầu tiên trong **s** /NULL/
int strcmp(s1,s2)	so sánh **s1** và **s2**; rtn **<0**, **0**, **>0** nếu **s1<s2**, **=s2**, **>s2**
char *strcpy(s1,s2)	sao chép **s2** đến **s1**; rtn **s1**
size_t strcspn(s1,s2)	tìm **s1[i]** đầu tiên giống phần tử bất kỳ của **s2**; rtn **i**
char *strerror(n)	rtn ptr đến chuỗi báo lỗi tương ứng errorcode **n**
size_t strlen(s)	chiều dài của **s** (không tính NULL)
char*strncat(s1,s2,su)	nối tối đa **su** ký tự từ **s2** vào cuối **s1**; rtn **s1**
int strncmp(s1,s2,su)	so sánh tối đa **su** ký tự đầu **s1** với **s2** (xem **strcmp**)
char *strncpy(s1,s2,su)	sao chép tối đa **su** ký tự từ **s2** đến **s1**; nếu s2 ngắn hơn **su**; các byte null sẽ nối tiếp sau; rtn **s1**
char *strpbrk(s1,s2)	tìm **s1[i]** đầu bằng phần tử bất kỳ của **s2**; rtn **&s1[i]**
char *strrchr(s,c)	rtn ptr đến ký tự **c** xuất hiện cuối trong **s** /NULL/
size_t strspn(s1,s2)	tìm **s1[i]** đầu tiên khác các phần tử của **s2**; rtn **i**
char *strstr(s1,s2)	rtn ptr đến chuỗi con đầu tiên trong **s1** trùng với **s2**
char *strtok(s1,s2)	tách **s1** thành các token phân cách bởi các ký tự trong **s2**; từ lần gọi thứ hai **s1**=NULL; **s2** có thể khác mỗi lần gọi; rtn ptr đến token hoặc NULL

Thời gian — time.h

char *asctime(*tm)	chuyển **struct tm** thành chuỗi; rtn ptr đến chuỗi đó
clock_t clock()	CPU time trong 1.0/CLOCKS_PER_SEC từ khi chương trình khởi động /-1/
char *ctime(*tl)	chuyển time ptd **tl** thành chuỗi; rtn ptr đến chuỗi đó
double difftime(tl1,tl2)	time chênh lệch **tl1-tl2** tính bằng giây
struct tm *gmtime(*tl)	chuyển time trỏ bởi **tl** thành time quốc tế (GMT)
struct tm*localtime(*tl)	chuyển time trỏ bởi **tl** thành time địa phương
time_t mktime (struct tm *tptr)	thay đổi **tptr** để thể hiện một mã hóa tương đương time địa phương /-1/
size_t strftime(s1,su, s2, struct tm *tptr)	ghi **tptr** đến vùng đệm **s1** với định dạng **s2**; kích thước vùng đệm là **su**; rtn số ký tự lưu được /0/
time_t time(*tl)	trả về time & date tính ra giây; nếu **tl<>NULL**, time được lưu trong ***tl**; chuyển đổi time được trả về với **ctime**, **localtime** hoặc **gmtime** /-1/

Danh sách đối số có chiều dài thay đổi — stdarg.h

***type* va_arg (ap,*type*)**	lấy đối số tiếp trong **ap** (kiểu **va_list**); **ap** phải khởi tạo bởi **va_start**; kiểu đối số phải là ***type***
void va_end(ap)	kết thúc danh sách đối số thay đổi
void va_start(ap,*pN*)	khởi tạo danh sách đối số thay đổi **ap**; ***pN*** là đối số đứng trước **(...)** trong nguyên mẫu hàm

CÁC ĐỐI SỐ DÒNG LỆNH

Các đối số nhập tại dòng lệnh được chuyển cho chương trình thông qua:
argc là số lượng các đối số +1;
argv là một mảng các con trỏ ký tự chỉ đến từng đối số.
argv[0] chỉ đến chuỗi tên chương trình đang thực hiện (kể cả path).
argv[argc] chứa con trỏ **NULL**.
Dùng **sscanf** chuyển các đối số lưu trong **argv** thành kiểu dữ liệu khác.
VÍ DỤ: Dùng chương trình **check** tại dòng lệnh: **check phone 35.79**
argc=3, argv[0]="check", argv[2]="35.79", argv[1]="phone", argv[3]=NULL
Để chuyển đổi số trong **argv[2]**, dùng **sscanf.**
VÍ DỤ: **int main (int argc, char *argv[ ])**
{ float amount; ... sscanf (argv[2],"%f",&amount); ... }