

Turbo C 2.0

· HOPE

运行库函数 源程序与参考大全

中国科学院希望高级电脑技术公司

TP312

X39

354123

1A

Turbo C 2.0 运行库函数源程序 与参考大全

萧 黎 尤晓东 金利群 编



中国科学院希望高级电脑技术公司

版权所有

不许翻印

违者必究

■ 北京市新闻出版局

准印证号： 3154 — 90154

■ 订购单位：北京 8721信箱资料部

■ 电 话： 2562329

■ 电 传： 01— 2561057

■ 电 挂： 0755

■ 地 址：海淀影剧院北侧

■ 乘 车： 320、332、302路海淀黄庄下车

■ 办公地点：公司大楼 101房间



Turbo C 2.0 运行库源程序与参考大全

《Turbo C 2.0 运行库源程序与参考大全》整理编译了 Turbo C 2.0 运行库中所有函数以及详细的函数使用说明。它不仅全部提供了《Turbo C 参考手册》中提供的函数，而且还提供了实现这些函数的所有低层支持函数。这使 Turbo C 2.0 运行库成为真正的开放式结构，用户可以从多方面高效地使用库函数。

不仅如此，有了库函数源程序，用户就可以对库函数进行单步断点等符号调试，可以领会 Turbo C 2.0 运行库以及整个软件的设计策略和技巧，为将来开发语言系统提供实例模型。Turbo C 之所以能风靡当今 PC 世界，不仅因为它有良好的用户界面，强大的符号调试功能，还因为有设计精良、运行高速、结构优化的运行库函数。运行库函数源程序让用户能目睹其真正风采，领略其库设计策略、技巧、程序设计风格、函数依赖关系等等。

有了库函数源程序，就可以无限制地把 Turbo C 与 Turbo Pascal 揉合起来，综合 C 与 Pascal 的优点编程，使程序设计“更上一层楼”。(关于具体的接口方法可与相榷，联系地址：北京航空航天大学 1-54 信箱李振格)。

对于高级程序员来说，通过修改库函数源程序，可更高效地、多方位地使用 Turbo C，对于一般的编程者和初学 C 的人，它又是一个很好的实例教材，是进行程序设计的良师益友。

齐全的运行库函数与低层支持函数，合《Turbo C 2.0 用户手册》、《Turbo C 2.0 参考手册》，将给用户带来无穷无尽的新发现与高效率。

本书在许多同志的帮助下，经过认真整理而成，由于篇幅浩大，源程序很多，虽然经过认真校对但错误难免，欢迎指正。

编者

一九九零年八月

目录

前言

Turbo C 库函数源程序

abortb	1	CheckOpenType	28
abs	1	_chmod	30
absread2abswrite	3	chmod	30
access	3	chsize	31
acos	4	circle	33
ACosAsin	4	_clear	87
allocmem	6	cleardevice	34
_AppendCtIZ	7	clearerr	34
arc	7	clearviewport	34
asctime	8	_close	34
asin	9	close	34
assert	9	closegraph	35
atan	9	clock	35
atan2	9	clreol	36
atexit	11	clrscr	36
atof	11	comtome	36
atoi	12	_control	37
atol	12	Copylt	38
bar	13	CopyUpr	39
bar3d	14	Coreleft	39
bdos	14	cos	40
bdosptr	15	cosh	40
bios	15	country	42
bioscom	16	cprintf	42
biosdisk	17	cputs	42
biosquip	19	_creat	43
bioskey	20	creat	43
biosmemory	21	CreatFile	45
biostime	22	creatnew	45
_brk	23	_crtinit	46
_brk	23	cscanf	46
brk	23	ctime	47
bsearch	24	ctrlbrk	48
_crtinit	24	delay	48
cas	26	delline	53
cd/loc	26	detecthgraph	53
ceil	26	diffime	53
_gets	27	disable	53
chdir	27	div	54
CheckFile	28	Displcement	54

_DOScmd	55	fillellipse	87
dosCreat	55	fillpoly	87
DOSenv	56	findfirst	87
dosexterr	57	findnext	89
dosReadOne	57	floodfill	89
dosSeekFinalChar	58	floor	89
_DOStimeToU	58	flushall	90
dostounix	60	FlushOutStreams	90
doswriteNone	61	fmod	91
DotFound	61	fnmerge	92
drawpoly	61	fnsplit	93
dup	62	fopen	94
ecvt	63	_fpreset	95
egainstalled	63	FP_OFF	95
ellipse	64	FP_SEG	96
emit	64	fprintf	96
enable	64	_fputc	96
dof	64	fputc	96
Exchange	65	fputchar	97
exec...	66	_fputn	98
exit	73	fputs	98
exit	73	fread	98
exp	73	free	99
fabs	75	freemem	101
farcalloc	76	freopen	101
farcoreleft	76	frexp	101
farfree	76	fscanf	102
farmalloc	78	fseek	102
farrealloc	80	fsetpos	103
fclose	81	fstat	103
fcloseall	81	ftell	105
fcvt	82	ftime	105
fdopen	82	fwrite	106
feof	82	gcvt	106
ferror	82	geninterrupt	106
fill	83	Get	107
iflush	83	getarccoords	107
fgetc	84	getaspectratio	107
fgetchar	85	getbkcolor	107
fgetc	85	getc	108
fgetpos	86	getchbrk	110
fgets	86	getch	111
filelength	86	getchar	111
fileno	87	getchc	111

getcolor	111	grapherrormsg	135
getcurdir	112	_graphexit	135
getcwd	112	_graphfreemem	135
getdate	113	_graphfreemem	135
getdefaultpalette	114	graphresult	136
getdfree	114	harderr	137
getdisk	115	hardresume	138
getdrivename	115	hardretn	138
getdta	115	_heaplen	138
getenv	116	hentry	139
getfat	117	Hex4	139
getfatd	118	highvideo	140
getfillpattern	119	hyperb	140
getfillsettings	119	hypot1	40
getfp	120	imagesize	141
getftime	120	initgraph	141
getgraphmode	121	inp	144
getimage	122	import	144
GetIndex()	122	inline	144
getlinesettings	122	installuserdrevr	145
getmaxcolor	123	installuserfont	145
getmaxx	124	Int0Catcher()	146
getmaxy	124	Int4Catcher()	146
getmodename	124	Int5Catcher()	146
getmoderange	124	Int6Catcher()	147
getpalette	124	int86	148
getpass	126	int86x	148
getpixel	126	intdos	150
getpsp	127	intdosx	150
gets	127	intr	151
getswitchar	128	ioctl	153
gettext	128	_IOerror	154
gettextinfo	128	is...	156
gettextsettings	129	isatty	157
gettime	131	isDST	157
getvect	131	itoab	157
getverify	132	_KbdFlu	159
getviewsetting	132	kbhit	159
getw	133	keep	160
getx	133	labs	160
gety	133	ldexp	160
gmtime	133	ldiv	161
gotoxy	134	ldtrunc	162
graphdefaults	134	lfind	164

line	164	peekb	192
linere1	164	perror	193
lineto	164	pieslice	193
localtime	165	poke	194
lick	165	pokeb	194
log	166	poly	194
log10	166	pow	195
longjump	167	pow10	198
lowvdo	168	...printf	200
_lrotl	168	putc	205
_lrotr	169	putch	205
_lsearch	169	purcha	206
lsearch	170	putenv	206
lseek	170	putimage	208
ltoa	171	putpixel	208
malloc	171	puts	208
lsetmem	173	puttext	209
_matherr	174	putw	209
matherr	175	qsort	209
max	177	qSortHelp	210
mem...	177	ruise	211
mkdir	180	rand	212
MK_FP	181	randbrd	212
_mkname	181	randbwr	213
mktemp	181	random	213
modf	182	randomize	214
movedata	183	_read	214
moverel	183	read	214
movetext	183	_realcvt	216
moveto	184	realloc	221
movemem	185	rectangle	221
normalize	186	registerbgidriver	221
normvideo	186	remove	222
nosound	186	rename	222
_open	187	restorecrtmode	223
open	187	rewind	223
_openfp	190	rmdir	223
outp	190	_rotl	223
outport	190	_rotr	224
outportb	191	_shrk	224
outtext	191	_sbrk	225
outtextxy	191	sbrk	225
parsfnum	191	...scanf	225
peek	192	_scanner	231

scanpop	244	setwritemode	268
scanrslt	244	signal	268
scantod	245	sin	270
_scantol	251	sinh	270
_screenio	255	sleep	271
screenpos	255	sopen	272
_scroll	255	sound	272
_searchpath	256	spawn...	272
searchpath	257	sprintf	276
sector	257	sqrt	276
segread	258	srand	277
setactivepage	258	ssanf	277
setallpalette	259	stat	277
setaspectratio	259	_status87	280
setbkcolor	259	stime	280
setblock	259	_stklen	280
setbuf	260	sd_Digit	280
setchrk	260	stpcpy	281
setcolor	261	Str...	282
setdate	261	_strerror	296
setdisk	261	strerror	296
setdta	261	strputn	297
setfillpattern	262	strtoul	297
setfillstyle	262	swab	298
setftime	262	system	298
setgraphbufsize	262	tan	299
setgraphmode	263	tanh	300
setjmp	263	tell	301
setlinestyle	264	textattr	301
setmem	264	textbackground	302
setmode	264	textcolor	303
setpalette	265	textheight	303
setrgbcolor	265	textmode	304
setgbpalette	265	textwidth	305
setswitchar	265	time	305
settextjustify	266	tmpfile	305
settextstyle	266	tmpnem	305
settime	266	toascii	306
setusercharsize	266	_tolower	306
setvbuf	267	tolower	306
setvect	267	_toupper	307
setverify	268	toupper	307
setviewport	268	trig	307
setvisualpage	268	_TrimCtlZ	308

TrimTrailing	308
tzset	309
ultoa	310
umask	310
UnGet	311
ungetc	311
ungetch	311
unixtodos	311
unlink	312
unlock	313
va...	313
va_arg	314
va_end	314
_validatexy	314
wva_start	314
vfprintf	315
vfscanf	315
_VideoInt	315
_vprinter	316
vprintf	330
vptr	331
_vram	331
vscanf	332
vsprintf	332
vsscanf	333
wherex	333
wwherey	333
window	334
_write	334
write	335
_xclose	336
_xcvt	336
_xfclose	340
_xflush	341
zapline	341

Turbo C 库函数源程序

本章按字母顺序列出 Turbo C 2.0 软件包中提供的各个库函数, 介绍它们的功能、用法、相关函数用法、原型所在头文件、说明信息、返回值、可移植性、参阅部分和示例, 并给出实现该函数的源程序, 具体格式如下:

函数名	功能
用法	本函数的用法
相关函数	
用法	本函数的相关函数的用法
原型在	本函数原型所在的头文件
说明	本函数的使用信息说明
返回值	本函数的返回值
可移植性	本函数的适用系统
参见	和本函数有关的其它函数名
示例	本函数使用的程序例
源程序	实现本函数的源程序

abort 异常终止—进程

用法	<code>void abort(void);</code>
原型在	<code>stdlib.h</code> , <code>process.h</code>
说明	本函数写终止信息到 <code>stderr</code> 中, 并通过 <code>_exit</code> 调用终止程序执行。
返回值	本函数不返回值
可移植性	适用于 UNIX 系统
参见	<code>assert</code> , <code>_exit</code> , <code>exec...</code> , <code>exit</code> , <code>spawn...</code>

abs 绝对值

—`abs.cas`

用法	<code>int abs(int i);</code>
相关函数	<code>double cabs(struct complex znum);</code>
用法	<code>double fabs(double x);</code> <code>long labs(long n);</code>
原型在	<code>stdlib.h</code> , <code>math.h</code>
说明	本函数返回整数参数 <code>i</code> 的绝对值, 如果包含了 <code>stdlib.h</code> 头文件并调用了 <code>abs</code> , <code>abs</code> 将作为可扩展为插入代码的宏看待。如果不包含 <code>stdlib.h</code> (或包含了但使用了 <code>#undef abs</code>), <code>abs</code> 将作为函数看待而不是宏。 <code>cabs</code> 是一计算复数 <code>znum</code> 绝对值的宏。 <code>znum</code> 是一具有类型 <code>complex</code> 的结构。此结构在 <code>math.h</code> 定义如下:

```
struct complex{
    double x,y;
};
```

调用 `cabs` 等价于调用具有 `znum` 的实部和虚部的 `sqrt`, 如下式所示:

```
sqrt(znum.x*znum.x+znum.y*znum.y)
```

如果没有包含 `math.h` 头文件 (或包含了而又使用了 `#undef cabs`), `cabs` 将作为一函数看待而

不是宏。

fabs 计算 **double** 类型 **x** 的绝对值。

labs 计算 **long** 整型 **n** 的绝对值。

返回值 **abs** 返回 0 到 32767 间的一个整数，有一个例外：当参数值为 -32768 时，返回 -32768。

cabs 返回 **double** 型的 **znum** 的绝对值，如果发生溢出，将返回 **HUGE_VAL** 并置 **error** 为：

ERANG 结果超出范围

可以通过 **matherr** 函数修改对 **cabs** 的错误处理。

fabs 返回 **x** 的绝对值，**lass** 返回 **n** 的绝对值，它们都不返回出错信息。

可移植性 适用于 **UNIX** 系统

参 见 **matherr** 源程序

```
#pragma inline
#include <stdlib.h>
#undef abs /* abs is defined as a macro in stdlib.h */
int abs(int i)
{
    asm      mov     ax, i
    asm      or      ax, ax
    asm      jge     exit
    asm      neg     ax
    exit:
    return _AX;
}
```

absread 读数据

—absread.cas

用 法 **int absread(int drive,int nsects,int sectno,**
void *buffer);

相关函数

用 法 **int abswrite(int drive,int nsects,int sectno,**
void *buffer);

原 型 在 **dos.h**

说 明 这些函数读和写磁盘的指定扇区内容。它们忽略磁盘的逻辑结构，并不关心文件、FAT 和目录。

absread 通过 **DOS** 中断 **0x25** 读指定的磁盘扇区内容。

abswrite 通过 **DOS** 中断 **0x26** 写内容到指定的磁盘扇区。

drive=要读的磁盘驱动器号(0=A,1=B 等)

nsects=要读的扇区数

sectno=要读的起始扇区号

buffer=数据读或写的存储区地址

读的扇区数受段中 **buffer** 以上的存储空间量的限制。因此，在一次调用 **absread** 或 **abswrite** 过程中，最大的可读存储空间为 64k 字节。

返回值 如果调用成功，两函数都返回 0。

如果调用出错，都返回 -1，并置 **error** 为经系统调用后返回 **AX** 的寄存器的值。关于 **error** 的说明，请参阅 **MS-DOS** 使用手册。

可移植性 只适用于 **MS-DOS**

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
#include <errno.h>
```

```

#pragma warn -use
int absread(int drive, int nsects, int lsect, void *buffer)
{
    register bogus1, bogus2;          /* force save of SI and DI */
    asm    mov    al,drive
    asm    mov    cx,nsects
    asm    mov    dx,lsect
    pushDS
    asm    LDS    _bx,buffer
    asm    int    25h
    asm    pop    bx                    /* clear old flags */
    popDS
    asm    jc     abserr
    return(0);
abserr:
#ifdef _HUGE_
    asm    mov    bx,seg errno
    asm    mov    es,bx
    asm    mov    es:errno,ax
#else
    asm    mov    errno,ax
#endif
    return(-1);
}
#pragma warn .use

```

abswrite 写数据

—absread.cas

用 法 int abswrite(int drive,int nsects,int sectno,
void *buffer);

原 型 在 dos.h

说 明 见 absread

源 程 序

```

#pragma warn -use
int abswrite(int drive, int nsects, int lsect, void *buffer)
{
    register bogus1, bogus2;          /* force save of SI and DI */
    asm    mov    al,drive
    asm    mov    cx,nsects
    asm    mov    dx,lsect
    pushDS
    asm    LDS    _bx,buffer
    asm    int    26h
    asm    pop    bx                    /* clear old flags */
    popDS
    asm    jc     abserr
    return(0);
abserr:
#ifdef _HUGE_
    asm    mov    bx,seg errno
    asm    mov    es,bx
    asm    mov    es:errno,ax
#else
    asm    mov    errno,ax
#endif
    return(-1);
}
#pragma warn .use

```

access 确定文件的存取权限

—access.c

用 法 int access (char *filename,int amode);

原 型 在 io.h

说 明 本函数检查给定名的函数是否存在，它的可读、可写和可执行性。filename 是指向文件名字符串的指针。

包含在 amode 中的们模式结构如下：

- 06 检查读/写允许
- 04 检查读允许
- 02 检查写允许
- 01 执行(被忽略)
- 00 检查文件的存在性

注意：在 MS-DOS 下，所有存在的文件具有读访问(amode=04)性，所以 00 和 04 有同样的结果。同理，06 和 02 是等价的，因为在 MS-DOS 下，写访问性隐含了读访问性。

如果 filename 指向一目录，access 就只确定该目录是否存在。

返回 值 如果要求的存取是允许的，返回 0；否则返回-1，并置 error 为以下值之一：

- ENOENT 路径或文件名没找到
- EACCES 无此权限

可移植性 适用于 UNIX 系统

参 见 chmod

源 程 序

```
#include <io.h>
#include <dos.h>
#include <errno.h>
int access(const char *filename, int amode)
{
    register int attrib;
    attrib = _chmod(filename, 0);
    if (attrib == -1)
        return(attrib);
    if ((amode & 0x00:2) == 0 || (attrib & FA_RDONLY) == 0)
        return(0);
    errno = EACCES;
    return(-1);
}
```

acos 三角函数

—acosasin.c25

用 法 double acos(double x);

原 型 在 math.h

说 明 trig

源 程 序

```
double acos (double x)
{
    return AcosAsin ("acos", 0xFF, &x);
}
```

AcosAsin 计算反正弦或反余弦。

—acosasin.cas

用 法 double AcosAsin (char *whoS, bits16 flags, double *xP),

说 明 计算由 xP 指向的数的反正弦或反余弦值。

返回 值 返回 xP 的反正弦或反余弦值。

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <rules.h>
#include <math.h>
```

```

#include < math.h> #include <errno.h>
static unsigned short NANINVRTIG [4] = {0,0,0x0440, 0x7FF0};
#pragma warn -rvl
#pragma warn -use
static double AcosAsin (char *whoS, bits16 flags, double *xP)
{
    bits16 status;
    int temp;
    register SI, DI; /* prevent the compiler making its own usage */
    asm mov dx, flags
    asm mov si, W0 (xP)
    asm mov cx, SS [si+6]
    asm and BY0 (SS [si+7]), 07Fh /* absolute value */
    asm fld DOUBLE (SS [si])
    asm shl cx, 1
    asm rcr dh, 1 /* DH = sign bit */
    asm jcxz acs_zero
    asm cmp cx, 7FE0h /* biased exponent of 1.0 */
    asm jnb acs_extreme
    /*
    Use the trig identity asin (x) = atan (x / sqrt (1 - x^2)).
    */
    asm FLDI
    asm FLD ST(1) /* duplicate X */
    asm FMUL ST(0), ST(0) /* square it */
    asm FSUBP ST(1), ST /* 1 - X^2 */
    asm FSQRT
    /*
    We now have tangent = ST(1)/ST. In order to use the FPATAN instruction
    we must order them so that ST(1) < ST. The special case of equality
    (angle pi/4) must be treated separately.
    */
    asm FCOM
    asm FSTSW W0 (status)
    asm mov ah, 41h /* C3, C0 are the important bits */
    asm FWAIT
    /*
    At this stage note that acos (x) = atan (sqrt (1-x^2) / x), which is
    the inverse of the tangent ratio used for asin (x). That is why
    DL, the inversion flag, started as OFF for acos, and 0 for asin.
    */
    asm and ah, BY1 (status)
    asm jz acs_atan
    asm add ah, ah
    asm js acs_pi_4
    asm FXCH
    asm not dl /* remember the exchanged order */ acs_atan:
    asm FPATAN /* arctan (ST(1) / ST) */
    asm or dl, dl /* should ratio be inverted ? */
    asm jz acs_sign
    asm mov W0 (temp), -1
    asm FILD W0 (temp)
    asm FLDPI
    asm FSCALE /* pi/2 */
    asm FSTP ST(1)
    asm FSUBRP st(1), st /* ST = pi/2 - ST */
    acs_sign:
    asm or dh, dh
    asm jns acs_end
    asm FCHS
    asm cmp BY0 (flags), 0FFh /* are we doing acos () ? */
    asm jae acs_end
    asm FLDPI
    asm FADDP ST(1), ST
    acs_end:
    return;
    /*
    Special cases.
    */

```

```

acs_extreme:
asm    ja      acs_domain
asm    mov     ax, SS_ [si+4]      /* check for an exact value +- 1.0 */
asm    or      ax, SS_ [si+2]
asm    or      ax, SS_ [si]
asm    jnz     acs_domain
asm    jmp     short acs_one
acs_zero:
asm    mov     dh, 0                /* make zero unsigned. */
asm    FSTP    ST(0)                /* pop stack */
asm    cmp     BY0 (flags), 0FFh    /* are we doing acos() ? */
asm    je      acs_resultZ
acs_resultZ:
asm    FLDZ
asm    jmp     short acs_sign
acs_one:
asm    FSTP    ST(0)                /* pop stack */
asm    cmp     BY0 (flags), 0FFh    /* are we doing acos() ? */
asm    je      acs_resultZ
acs_resultP2:
asm    mov     W0 (temp), -1 asm    FILD    W0 (temp)
asm    FLDPI
asm    FSCALE
asm    FSTP    ST(1)                /* pi/2 */
asm    jmp     short acs_sign
acs_pi_4:
asm    FCOMPP                        /* pop two items off the stack */
asm    mov     W0 (temp), -2
asm    FILD    W0 (temp)
asm    FLDPI
asm    FSCALE                        /* pi/4 */
asm    FSTP    ST(1)
asm    jmp     short acs_sign
/*
If the argument was outside [-1,+1] then it is a domain error.
*/
acs_domain:
asm    or      BY0 (SS_ [si+7]), dh /* put the sign back */
asm    FSTP    ST (0)              /* pop x from stack */
#pragma warn .ret
return _matherr (DOMAIN, whoS, xP, NULL, *((double *) NANINVTRIG));
#pragma warn .ret
}
#pragma warn .rvl
#pragma warn .use

```

allocmem 分配 DOS 存储段

—allocmem.cas

用 法 int allocmem(unsigned size,unsigned *seg);

相关函数 int freemem(unsigned seg);

用 法 int setblock(int seg,int newsiz);

原 型 在 dos.h

说 明 本函数使用 MS-DOS 系统调用 0x48 来分配自由存储块，并返回分配块的段地址。

size 是段中要求的存储空间大小。seg 是一指针，它指向将被赋值的新分配块的段地址的地址字。如果没有足够的空间可供分配，将不对 seg 所指的地址字赋值。

所有被分配的块是用指针连在一起的段。

freemem 释放以前用 allocmem 调用所分配的存储块。seg 是该块的段地址。

setblock 修改存储段的大小。seg 是以前调用 allocmem 后返回的段地址。newsiz 是新的、要求的段中存储空间大小。

返 回 值 如果调用成功，allocmem 返回-1。若出错，将返回-1数(为最大可用块大小)。

如果调用成功，freemem 返回 0，否则返回-1，并置 error 为：

ENOMEM 无足够存储空间

如果调用成功, setblock 返回值-1, 否则返回最大可用块的大小。

allocmem 或 setblock 返回的错误码将置-doserrno 和全局变量 error 为:

ENOMEM 无足够的存储器

可移植性 只适用于 MS-DOS

参 见 malloc

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
#include <io.h>
int allocmem(unsigned size, unsigned *segp)
{
    asm    mov    ah, 48h
    asm    mov    bx, size
    asm    int    21h
    asm    jc     allocmemFailed
    asm    LES    di, segp
    asm    mov    ES, [di], ax
    return(-1);
allocmemFailed:
    asm    push    bx
    asm    _IOerror(_AX);
    asm    pop     ax
    return (_AX);
}
```

函 数 名 __AppendCtlZ - 给文件添加 Ctrl-Z。

—zapctrlz.cas

用 法 void pascal __AppendCtlZ (int fildes)

原型文件 _io.h

说 明 给文件添加一个 Ctrl-Z。

源 程 序

```
void pascal __AppendCtlZ (int fildes)
{
    char c = _ctlZ;
    dosSeekFinatChar (fildes);
    if (dosReadOne (fildes) != _ctlZ)
    {
        pushDS
        asm    mov    bx, fildes
        asm    mov    cx, 1
        #if LDATA
        asm    push    SS
        asm    pop     DS
        #endif
        asm    lea    dx, c
        asm    mov    ah, 40h
        asm    int    21h /* dosWrite (fildes, DS:DX, CX) */
        popDS
    }
}
```

arc 画一条圆弧

用 法 #include <graphics.h>

```
void far arc(int x,int y,int stangle,int endangle,
int radius);
```

相关函数 void far circle(int x,int y,int radius);

```

用 法 void far ellipse(int x,int y,int stangle,int endangle
      int xradius,int yradius);
void far getarccoords(struct arccoords type,
                      far *arccoords);
void far getspectration(int far *xasp,int far *yasp);
void far pieslice(int x,int y,int stangle,int endangle
                  int radius);

```

说 明 此处描述四个画线函数(arc、circle、ellipse和pieslice)用当前颜色画出图形的外廓。
 arc以(x,y)为圆心, radius为半径, 从起始角stangle到终止角endangle画一圆弧。如果stangle=0且endangle=360, 则对arc的调用将画一整圆。
 circle以(x,y)为圆心, radius为半径画一圆。
 ellipse是(x,y)为中心画一椭圆, 其长短轴由xradius和yradius给出, 从起始角stangle画到终止角endangle。如果stangle=0且endangle=360, 则对ellipse的调用将画出一完整的椭圆。
 pieslice以(x,y)为中心, radius为半径, 从起始角stangle到终止角endangle画并填充一扇形。pieslice用当前的画线颜色画出扇形的外廓, 然后用当前填充模式和填充颜色填充。
 arc、ellipse和pieslice的角度是逆时针方向的, 0度在3点, 90度在12点等。
 每一个图形驱动器和图形方式都有一关联的纵横比。
 arc、circle和pieslice函数用纵横比作为比例因子以保证在屏幕的圆呈圆形。该纵横比可通过调用getspectratio函数来计算, 可用它处理*xasp和*yasp。
 Y纵横因子*yasp一般为10000, 除了VGA, 在所有的图形适配器上, *xasp(X纵横因子)都比*yasp小, 因为像素的高度总大于宽度。在像素是“方形”的VGA上, *xasp=*yasp。
 一般说来, *yasp与*xasp之间的关系大致可表示为:

*yasp=10000

*xasp<=10000

getarccoords函数将有关上次调用arc的信息填入到arccoords所指的arccoordstype结构中。arccoordstype结构在graphics.h中定义如下:

```

struct arccoordstype{
    int x,y;
    int xstart,ystart,xend,yend;
};

```

结构中的成员用来说明arc的中心点(x,y)、起始位置(xstart,ystart)以及终止位置(xend,yend)。如果需要在arc函数所画的末尾画一条线, 这些信息十分有用。

返回 值 如果填充扇形时发生错误, graphresult函数返回-6。

可移植性 在Turbo Pascal 5.0中有相似的程序

参 见 getfillsettings

asctime 转换日期和时间为ASCII码

—ctime.c

用 法 #include <time.h>

char *asctime(struct tm *tm);

原 型 在 time.h

说 明 ctime

源 程 序

char *asctime(const struct tm *tm)

```

{
    static char    a[26];
    sprintf(a, "%s %s %02d %02d:%02d:%02d %4d\n", Weekday[tm->tm_wday],
Months[tm->tm_mon], tm->tm_mday, tm->tm_hour, tm->tm_min,
tm->tm_sec, tm->tm_year + 1900);
    return(a);
}

```

asin 三角函数

—acosasin.cas

用 法 double asin(double x); 原型在 math.h

说 明 见 trig

源 程 序

```

double asin (double x)
{
    return AcosAsin ("asin", 0x00, &x);
}

```

assert 测试一个条件并可能使程序终止

用 法 #include <assert.h>

void assert(int test);

原 型 在 assert.h

说 明 assert 是一个可扩展为 if 语句的宏。如果扩展宏中的测试失败，将输出一信息并通过调用 abort 中止程序。输出的信息为：

Assertion failed:file filename,line linenum

其中: filename 和 linenum 是源文件名和 assert 宏出现的行号。

如果在源文件代码的#include <assert.h> 指令之前放上#define NDEBU 指令，其作用是对 assert 进行注释。

可移植性 本宏适用于某些 UNIX 系统，如系统 III 和系统 V。

参 见 abort

atan 反正切三角函数

—atan.cas

用 法 double atan(double x);

原 型 在 math.h

说 明 见 trig

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <rules.h>
#include <math.h>
#include <_math.h>
#pragma warn -rvl
double atan (double x)
{
    asm      FLD     DOUBLE (x)
    asm      _FAST_  (_FATAN_)
    return;
}
#pragma warn .rvl

```

atan2 三角函数

—atan2.cas

用 法 double atan2(double y,double x);

原 型 在 math.h

说 明 见 trig

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#include <errno.h>
static unsigned short piBy2 [] = {0xC235, 0x2168, 0xDAA2, 0xC90F, 0x3FFF}; static unsigned
short NANINVTRIG [4] = {0,0,0x0440, 0x7FF0};
#pragma warn -rvl
#pragma warn -use
double atan2 (double y, double x)
{
    register SI, DI; /* prevent the compiler making its own usage */
    asm FLD DOUBLE (x)
    asm mov ax, x [6] /* select MSW of x */
    asm and al, 0F0h /* ignore fraction */
    asm mov bx, y [6] /* .. and of y */
    asm and bl, 0F0h
    asm shl bx, 1 /* discard sign */
    asm FLD DOUBLE (y)
    asm jz at2_yIsZero
    asm shl ax, 1 /* discard sign */
    asm jz at2_xIsZero
    asm cmp ax, 0FFE0h
    asm jnb at2_xIsInf
    asm cmp bx, 0FFE0h
    asm jnb at2_yIsInf
    asm FDIVRP ST(1), ST(0)
    asm FAST (_ATAN_)
    /* convert the simple answer to a four quadrant answer. */
    at2_setQuad:
    asm test BY0 (x [7]), 80h /* the sign bit */
    asm jz at2_end
    asm FLDPI
    asm test BY0 (y [7]), 80h
    asm jz at2_2ndQuad
    at2_3rdQuad:
    asm FSUBP ST(1), ST
    asm jmp short at2_end
    at2_2ndQuad:
    asm FADDP ST(1), ST
    at2_end:
    return;
    /* Special cases.
    */
    at2_yIsZero:
    asm shl ax, 1
    asm jz at2_indeterminate /* if both are zero */
    asm jc at2_retPi /* x < 0, return Pi */
    asm FSTP ST(1) /* else y is result */
    asm jmp short at2_end
    at2_retPi:
    /* y = 0, x < 0 */
    asm FCOMPP /* discard x and y */
    asm FLDPI /* and return Pi */
    asm jmp short at2_end
    at2_xIsZero:
    /* and y is not zero
    or
    and x is finite
    discard x and y
    */
    at2_yIsInf:
    /* check sign of Y */
    asm FCOMPP
    asm FLD tbyte ptr (piBy2)
    asm test BY0 (y [7]), 80h /* positive - return Pi/2 */
    asm jz at2_HPi /* negative - return -Pi/2 */
    asm FCHS
    at2_HPi:
    asm jmp short at2_setQuad
    at2_xIsInf:
    asm cmp bx, 0FFE0h
```

```

asm    jnb      at2_indeterminate      /* if both are infinite */
asm    FCOMPP                                /* discard x and y */
asm    FLDZ
asm    jmp      short    at2_setQuad.
/*
   There are two cases considered irresolvable: both operands zero, or
   both operands infinite.
*/
at2_indeterminate:                        /* either equal or both infinite */
asm    FCOMPP                                /* discard x and y */
#pragma warn -ret
        return _m2therr (DOMAIN, "atan2", &x, &y, *((double *) NANINVERTIG));
#pragma warn .ret
}
#pragma warn .rvl
#pragma warn .use

```

atexit 注册终止函数

—atexit.c

用 法 #include <stdlib.h>

int atexit(atexit_t func)

说 明 本函数用于注册由 func 指的函数为“退出(exit)函数”。在程序正常终止情况下，exit 在返回操作系统前调用 func(无参数)。被调用的函数是 atexit_t 型的，该类型在 stdlib.h 中的 typedef 中定义。

每次对 atexit 的调用注册另一个退出函数，最多可以注册 32 个函数，它们按后进先出的顺序执行。

返 回 值 如果调用成功，atexit 返回 0。否则返回非零值(没有足够空间注册函数)。

参 见 参 见 exec..., exit, spawn...

源 程 序

```

#include <stdlib.h>
#define MAX_ATEXIT      32
int    _atexitcnt = 0;      /* count of atexit functions */
atexit_t _atexittbl[MAX_ATEXIT]; /* array of atexit function pointers */
int atexit(atexit_t func)
{
    if (_atexitcnt == MAX_ATEXIT)
        return(1);
    _atexittbl[_atexitcnt++] = func;
    return(0);
}

```

atof 把一字符串转换成浮点数

—atof.c

用 法 double atof(char *nptr);

相关函数 int atoi(char *nptr);

用 法 long atol(char *nptr);

原 型 在 math.h, stdlib.h

说 明 本函数把由 nptr 所指的字符串转换成 double 类型，它识别：

- 一个任选的制表和空格字符串
- 一个任选符号
- 接着是一数字串和一个任选的十进制小数点
- 接着是任选的 e 或 E，后跟一个任选的带符号整数

atoi 把由 nptr 所指的字符串转换成 int 类型；atolten 把字符串转换成 long 型。它们识别：

- 一个任选的制表和空格字符串
- 一个任选符号

• 接着是一串数字

在所有这三种函数中, 如果遇到一个不认识字符, 则转换停止。

所有这三个函数都无溢出限定

返回值 这些函数返回输入字符串的转换值, 如果该字符串不能转换为对应类型的数(atof 为 double, atoi 为 int, atol 为 long)。则返回 0。

可移植性 适用于 UNIX 系统

参 见 scanf

源 程 序

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
double atof(const char *strP)
{
    return strtod(strP, NULL);
}
```

atoi 把字符串转换成整数

—atol.cas

用 法 int atoi(char *nptr);

原 型 在 stdlib.h

说 明 见 atof

源 程 序

```
int atoi (const char *strP)
{
    return (int) atol (strP);
}
```

atoi 把字符串转换成长整型数

—atol.cas

用 法 long atol(char *nptr);

原 型 在 stdlib.h

说 明 见 atof

源 程 序

```
#pragma inline #include <asmrules.h>
#include <stdlib.h>
#include <ctype.h>
#undef atoi /* macro in stdlib */
#pragma warn -rvl
#pragma warn -use
long atol (const char *strP)
{
    register SI, DI; /* prevent the compiler making its own usage */
#ifdef LDATA
asm push ES
#endif
asm push bp
asm LES si, strP
#ifdef _HUGE
asm mov ax, seg _ctype
asm mov DS, ax
#endif
asm cld
asm sub ax, ax
asm cwd /* default result in DX:AX = 0:0. */
asm mov cx, 10 /* decimal radix */
/* Skip any isspace (ctype) characters at the start. */
asm mov bh, 0
asm mov di, 1 + offset (ES, _ctype) /* avoid assuming DS */
```

```

atl_skipSpace:
asm    mov    bl, ES_ [si]
asm    inc    si
asm    test   BY0 ((bx+di)), _IS_SP          /* (1 + _ctype) [bx]    */
asm    jnz    atl_skipSpace
/* Remember if a negative sign is encountered */
asm    mov    bp, 0                          /* BP holds sign, 0 = positive */
asm    cmp    bl, '+'
asm    je     atl_signSeen
asm    cmp    bl, '-'
asm    jne    atl_inspectDigit
asm    inc    bp                            /* 1 = negative */
atl_signSeen:
/* accumulate digits in AX until more than 16 bits. */
atl_digitOnWord:
asm    mov    bl, ES_ [si]
asm    inc    si
atl_inspectDigit:
asm    cmp    bl, '9'
asm    ja     atl_end
asm    sub    bl, '0'
asm    jb     atl_end
asm    mul    cx
asm    add    ax, bx
asm    adc    dl, dh
asm    jz     atl_digitOnWord
asm    jmp    short atl_nextDigit
/* Accumulate digits in DX:AX, overflow may occur but is ignored
   (as per SVID norm).
*/
atl_digitOnLong:
asm    mov    di, dx                        /* copy DX to safety in DI    */
asm    mov    cx, 10
asm    mul    cx
asm    xchg    ax, di
asm    xchg    dx, cx
asm    mul    dx
asm    xchg    dx, ax
asm    xchg    ax, di                      /* function result should be in DX:AX */
asm    add    ax, bx
asm    adc    dx, cx
/* overflows are discarded; result is undefined */
atl_nextDigit:
asm    mov    bl, ES_ [si]
asm    inc    si
asm    cmp    bl, '9'
asm    ja     atl_end
asm    sub    bl, '0'
asm    jnb    atl_digitOnLong
/* the result may be signed. */
atl_end:
asm    dec    bp                            /* was '-' negative seen ?    */
asm    jl     atl_exit
asm    neg    dx
asm    neg    ax
asm    shb    dx, 0                        /* negate (DX:AX) */
atl_exit:
asm    pop    bp
#ifdef LDATA
asm    pop    ES
#endif
return;
}
#pragma warn _rvt
#pragma warn _use

```

bar 画一个条形图

用 法 `#include <graphics.h>` `void far bar(int left,int top,int right,int bottom);`

相关函数 `void far bar3d(int left,int top,int right,int bottom,`

用 法 `int depth,int topflag);`

原型在 `graphics.h`

说 明 `bar` 画一填充长方条形。该条形用当前填充模式和填充颜色填充。`bar` 并不画出条形外廓。如要画二维的条形图，可以用 `bar3d` 并使 `depth=0`。

`bar3d` 画一个三维的矩形条形图，然后用当前填充模式和填充颜色填充。条形的外廓用当前的线型和颜色画出。以象素为单位的条形深度由 `depth` 给出。

`topflag` 参数决定是否在条形图上放一个三维顶。如果 `topflag` 不等于零，则放一个顶，否则不放顶(使得几个条形图叠在一起成为可能)。

在这两个函数中，矩形的左上角和右下角分别由 `(left,top)` 和 `(right,bottom)` 给出。

要计算 `bar3d` 的典型深度，可取原条形图的 $1/4$ 宽度。如：

`bar3d(left,top,right,bottom,(right-left)/4,1);`

返回 值 本函数不返回值

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 数 `getbkcolor`, `getfillsettings`, `getlinestyle`, `graphresult`, `rectangle`

bar3d 画三维条形图

用 法 `#include <graphics.h>`

`void far bar3d(int left,int top,int right,int bottom,int depth,int topflag);`

原型在 `graphics.h`

说 明 见 `bar`

bdos MS-DOS 系统调用

—bdos.cas

用 法 `int bdos(int dosfun,unsigned dosdx,unsigned dosal);`

相关函数

用 法 `int bdosptr(int dosptr,void *argument,unsigned dosal);`

原型在 `dos.h`

说 明 这些调用提供了对许多 MS-DOS 系统调用的直接存取。关于每个系统调用的细节，请见《MS-DOS 程序员参考手册》。

需用整型参数的系统调用使用 `bdos`，而需用指针参数的系统调用使用 `bdosptr`。

对于小数据模式(微型、小型和中型)，`bdos` 和 `bdosptr` 相似。而在大数据模式(紧缩、大型和巨型)中，使用由指针作为调用参数的 `bdosptr` 系统调用是很重要的。

`dosfun` 在《MS-DOS 程序员参考手册》中定义。

在小数据模式中，`bdosptr` 的 `argument` 参数传给 `DX`；在大数据模式中，它给出 `DS:DX` 的值，供系统调用使用。

对于 `bdos` 调用，`dosdx` 是 `DX` 寄存器的值。

`dosal` 是 `AL` 寄存器的值

关于使用 `bdosptr` 的例子，参阅 `harderr`

返回 值 `bdos` 的返回值是由系统调用设置的 `AX` 的值。

`bdosptr` 调用成功时，返回值为 `AX` 的值，如果调用失败，则返回 -1，并设置 `errno` 和 `_doserrno` 的值。

可移植性 只适用于 MS-DOS

源 程 序

```
#pragma inline
#include <dos.h>
#pragma warn -use
int bdos(int dosfn, unsigned dosdx, unsigned dosal)
{
    register int SI, DI; asm mov ah, dosfn
    asm mov al, dosal
    asm mov dx, dosdx
    asm int 21h
    return(_AX);
}
#pragma warn .use
```

bdosptr MS-DOS 系统调用

—bdosptr.cas

用 法 int bdosptr(int(dosfun,void *argument,unsigned dosal),

原 型 在 dos.h

说 明 见 bdos

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
#include <io.h>
#pragma warn -use
int bdosptr(int cmd, void *arg, unsigned dosal)
{
    register int SI, DI;
    pushDS_
    asm mov ah, byte ptr cmd
    asm mov al, byte ptr dosal
    asm LDS_ dx, arg
    asm cld
    asm int 21h
    popDS_
    asm jc bdosptrFailed
    return(_AX);
bdosptrFailed:
    return _IOError(_AX);
}
#pragma warn .use
```

函 数 名 bios - 通过 bios 移动屏幕数据块。

—screen.c

用 法 static void near pascal bios(void far *dst, void far *src,
int len);

返 回 值 无。

源 程 序

```
static void near pascal bios(void far *dst, void far *src, int len)
{
    register unsigned char cell;
    unsigned dstpos, srcpos, cursorpos;
    int originalpos, dstvram, srcvram;
    cursorpos = originalpos = _wherexy();
    if ((dstvram = (FP_SEG(dst) == _video.displayptr.u.seg)) != 0)
        dstpos = screenpos(dst);
    if ((srcvram = (FP_SEG(src) == _video.displayptr.u.seg)) != 0)
        srcpos = screenpos(src);
    while (len--)
    {
        if (srcvram)
```

```

    {
        moveto(&srcpos, &cursorpos);
        BH = 0;
        AH = 8;
        _VideoInt();
        charcell = _AX;
    }
    else
        charcell = *((unsigned far *)src)++;
    if (dstvram)
    {
        moveto(&dstpos, &cursorpos);
        AX = charcell;
        BL = AH;
        CX = 1;
        BH = 0;
        AH = 9;
        _VideoInt();
    }
    else
        *((unsigned far *)dst)++ = charcell;
}
DX = originalpos;
BH = 0;
AH = 2;
_VideoInt();
}

```

bioscom I/O 通信

—bioscom.cas

用 法 int bioscom(int cmd,char byte,int port);

原 型 在 bios.h

说 明 bioscmd 在由 port 指定的 I/O 端口上执行各种 RS232 通信。

port 值为零表示 COM1, 1 表示 COM2, 等等。

cmd 的值为下例之一:

- 0 设置通信参数为byte值
- 1 按字节发送字符到通信线上
- 2 从通信线上接受一个字符
- 3 返回通信端口的当前状态

byte 为下列位的组合:

- 0x02 7个数据位
- 0x03 8个数据位
- 0x00 1个停止位
- 0x04 2个停止位
- 0x00 无奇偶检验
- 0x08 奇检验
- 0x18 偶检验
- 0x00 110波特
- 0x20 150波特
- 0x40 350波特
- 0x60 600波特
- 0x80 1200波特
- 0xA0 2400波特
- 0xC0 4800波特

0xE0 9600波特

例如, 给定 byte 值为 0xEB(0xE0|0x08|0x00|0x03), 将设置通信口为 9600 波特, 奇检验, 1 个停止位和 8 个数据位。

返回值 对所有的 cmd 值, 返回一 16 位整数, 其中高 8 位是状态位, 低 8 位随 cmd 值而变化。返回值的高 8 位定义如下:

第15位	超时
第14位	传送移位寄存器空
第13位	传送保持寄存器空
第12位	中断检测
第11位	框架错误
第10位	奇偶错误
第9位	溢出超越错误
第8位	数据就绪

当 cmd 的值是 2 时, 如果不发生错误, 返回值的低位为读入字节。如果发生了错误, 则至少有一高位被置位。所以若无高位被置位, 则在接收字节过程中没有错误。

当 cmd 的值是 0 或 3 时, 返回值的高 8 位设置如上所述, 低 8 位定义如下:

第7位	接收线信号检测
第6位	响铃指示
第5位	数据设置就绪
第4位	清除发送
第3位	8接收线信号检测
第2位	后沿响铃检测
第1位	8数据设置就绪
第0位	8清除发送

可移植性 本函数仅适用于 IBM PC 及兼容机上。

源程序

```
#pragma inline
#include <bios.h>
#pragma warn -rvl
int bioscom(int cmd, char byte, int port)
{
    asm    mov    ah, cmd
    asm    mov    al, byte
    asm    mov    dx, port
    asm    int    14h
}
#pragma warn .rvl
```

biosdisk 硬盘/软盘 I/O

--biosdisk.cas

用法 biosdisk(int cmd,int drive,int head,int track,
int sector,int nsects,void *buffer);

原型在 bios.h

说明 本函数使用中断 0x13, 把磁盘操作直接转给 BIOS。

drive 是表示磁盘驱动器的数: 0 代表第一个软驱, 1 表示第二个软驱, 2 表示第三个等等。对于硬盘驱动器, drive 的值为 0x80, 表示第一个硬驱, 为 0x81 表示第二个硬驱, 0x82 表示第三个等等。

对于硬盘, 指定物理驱动器而不是硬盘分区, 应用程序如果需要的话, 必须自行解释分区表信息。

cmd 指示待执行的操作。根据 cmd 的值，可以需要或不需要其它参数。下面列出对应于 IBM PC, XT 或 AT 可能的 cmd 值:

- 0 复位磁盘系统。强制磁盘驱动控制器进行硬复位，忽略其它参数。
- 1 返回最后一次磁盘操作的状态，忽略其它参数。
- 2 读一个或多个扇区内容到存储区。起始扇区号由 head, track 和 sector 给定，扇区数由 nsects 给出，数据以每扇区 512 字节读入缓冲区 buffer。
- 3 写存储区内容到一个或多个磁盘区中。起始扇区号由 head, track 和 sector 给定，扇区由 nsects 给出。数据从缓冲区 buffer 中以每扇区 512 字节写入。
- 4 验证一个或多个扇区，起始扇区由 head, track 和 sector 给定，扇区数由 nsects 给出。
- 5 格式化一个磁道。磁道号由 head 和 track 给定。buffer 指向一个将写到给定磁道上的扇区头表。关于这个表的说明和格式化操作，请参阅《IBM PC 技术参考手册》。

以下 cmd 值仅适用于 XT 或 AT:

- 6 格式化一个磁盘并设置坏扇区标志。
- 7 从指定磁道开始，格式化一个驱动器。
- 8 返回当前驱动器参数。驱动器信息返回在 buffer 的前 4 个字节中。
- 9 初始化驱动器特性。
- 10 进行长读。每扇区读入 512 字节加上额外 4 个字节。
- 11 进行长写。每扇区写入 512 字节加上额外 4 个字节。
- 12 进行磁盘查找。
- 13 交替磁盘复位。
- 14 读扇区缓冲区。
- 15 写扇区缓冲区。
- 16 测试指定驱动器是否准备好。
- 17 重新校准驱动器。
- 18 控制器 RAM 诊断。
- 19 驱动器诊断。
- 20 控制器的内部诊断。

返回值 这些操作返回一个由下列位组成的状态字节:

- 0x00 操作成功。
- 0x01 错误命令。
- 0x02 地址标志没找到。
- 0x04 记录没找到。
- 0x05 复位失败。
- 0x07 驱动器参数活动失败。
- 0x09 操作试图超过界限。
- 0x0B 检测到坏磁道标志。
- 0x10 磁盘读错误。
- 0x11 控制器失败。
- 0x20 查找操作失败。
- 0x40 附属设备不响应。
- 0x80 未定义的错误出现。
- 0xBB 有意义操作失败。

注意: 0x11 不是错误, 因为数据是正确的。它返回的值只是给应用程序一个自行决定的机会。

可移植性 只适用于 IBM PC 及其兼容机

源程序

```
#pragma inline
#include <asmrules.h> #include <bios.h>
int biosdisk(int cmd, int drive, int head, int track,
             int sector, int nsects, void *buffer)
{
    #if ! (LDATA)
        _ES = _DS;
    #endif
    asm    mov    ah, cmd
    asm    mov    al, nsects
    asm    LES    bx, buffer
    asm    mov    cx, track
    asm    shr    cx, 1
    asm    shr    cx, 1
    asm    and    cl, 0C0h
    asm    add    cl, sector
    asm    mov    ch, track
    asm    mov    dh, head
    asm    mov    di, drive
    asm    int    013h
    asm    cmp    BY0(cmd), 8
    asm    jne    BiosDiskEnd
    asm    mov    W0(ES_ [bx]), cx
    asm    mov    W1(ES_ [bx]), dx
BiosDiskEnd:
    return _AH;
}
```

biosequip 检查设备

--biosequ.cas

用法 int biosequip(void);

原型在 bios.h

说明 本函数返回一描述和系统相连的设备的整数, 它使用了 BIOS 中断 0x11。

返回值 返回值解释为位字段集合, 在 IBM PC 上有下列值:

第 15 位 打印机数目

第 14 位 打印机数目

第 13 位 连接的串行打印机

第 12 位 连接的游戏 I/O

第 11 位 RS232 端口数

第 10 位 RS232 端口数

第 9 位 RS232 端口数

第 8 位 非 DMA。当第 8 位为 0 时, 机器有 DMA; 当第 8 位为 1 时, 没有 DMA 如 PC Jr。

第 7 位 硬盘数

第 6 位 磁盘数

00=1 个驱动器

01=2 个驱动器

10=3 个驱动器

11=4 个驱动器, 仅当第 0 位是 1 时

第 5 位 初始化

第 4 位 视频模式
 00=未使用
 01=40x25 BW带彩色卡
 10=80x25 BW带彩色卡
 11=80x25 BW带单色卡

第 3 位 母板

第 2 位 RAM大小
 00=16k
 01=32k
 10=48k
 11=64k

第 1 位 浮点协处理器

第 0 位 从软盘启动

可移植性 只适用于 IBM PC 及其兼容机

源程序

```
#pragma inline
#include <asmrules.h>
#include <bios.h>
int biosequip(void)
{
  asm      int      11h
          return _AX;
}
```

bioskey 键盘接口

—bioskey.cas

用法 int bioskey(int cmd);

原型在 bios.h

说明 本函数使用 BIOS 中断 0x14 执行各种键盘操作。参数 cmd 确定实际的操作：

- 0 返回下一从键盘输入的字符。如果低8位非零，则为ASCII字符；如果低8位为零，则高8位为扩展键盘代码，在《IBM PC技术参考手册》中定义。
- 1 测试是否有可读的输入键。如果返回0，则表示没有；否则返回下一个输入键。键还保存，供下次参数cmd为0时bioskey调用返回。
- 2 请求当前移位状态。状态值由下列值相或而得到：

0x80 Insert切换
 0x40 Caps切换
 0x20 Num Lock切换
 0x10 Scroll Lock切换
 0x08 按下Alt
 0x04 按下Ctrl
 0x02 按下Left shift
 0x01 按下Right Shift

可移植性 只适用于 IBM PC 及其兼容机

源程序

```
#pragma inline
#include <asmrules.h>
#include <bios.h>
#define KbdBreakVector 1BH
#define DOS 21H
```

```

/*
  There is a condition where DOS can crash if Ctrl-Brk is hit when in
  an INT 16 call. We also redirect the INT 1B (keyboard break)
  vector to an IRET when we're in INT 16.
*/
int bioskey(int cmd) {
asm    PUSH    DS          /* Save DS */
asm    PUSH    ES          /* Save ES */
asm    JMP     SHORT next
/* Nested procedure for handling KBD break interrupts (1BH) */
asm    nullinterrupt      PROC FAR
asm                                IRET
asm    nullinterrupt      ENDP
/* ES:BX <- Old kbd break routine address */
next
asm    MOV     AH,35H       /* DOS get interrupt vector */
asm    MOV     AL,KbdBreakVector
asm    INT     DOS
/* DS:DX <- New (null) break routine address */
asm    PUSH    CS
asm    POP     DS
asm    MOV     DX,OFFSET nullinterrupt
/* Set Kbd-Break address to interrupt routine */
asm    MOV     AH,25H       /* DOS set interrupt vector */
asm    MOV     AL,KbdBreakVector
asm    INT     DOS
/* Do the BIOS keyboard call */
asm    MOV     AH, cmd
asm    INT     16h
asm    JNZ     keydone
asm    CMP     BYTE PTR (cmd), 1
asm    JNE     keydone
asm    XOR     AX,AX
keydone:
asm    XCHG    AX,CX        /* CX <- rval, need AX for INT 21s */
/* Restore the old break vector (still in ES:BX) */
asm    PUSH    ES          /* DS:DX <- ES:BX */
asm    POP     DS
asm    MOV     DX,BX
asm    MOV     AH,25H       /* DOS set interrupt vector */
asm    MOV     AL,KbdBreakVector
asm    INT     DOS
asm    POP     ES          /* Restore ES */
asm    POP     DS          /* Restore DS */
asm    XCHG    AX,CX        /* Get return value back in AX */
return _AX; }

```

biomemory 返回存储块大小

—biosequ.cas

用法 int biomemory

原型 在 bios.h

说明 本函数通过 BIOS 中断 0x12 调用返回存储块大小。

返回值 返回以 1K 为单位的存储块大小

可移植性 只适用于 IBM PC 及其兼容机

源程序

```

#pragma inline
#include <asmrules.h>
#include <bios.h>
int biomemory(void)
{
asm    int     12h
asm    return _AX;
}

```

biosprint 打印机 I/O

—biosprin.cas

用 法 int biosprint(int cmd,int byte,int port);

原 型 在 bios.h

说 明 本函数在由参数 port 指定的打印机上执行各种打印机功能。

port 值为 0 对应于 LPT1, 1 对应于 LPT2 等等。

cmd 的值为下列之一:

- 0 打字 byte 中的字符
- 1 初始化打印机端口
- 2 读打印机状态

byte 的值可以是 0 到 255。

返 回 值 这些操作的返回值是下列位值通过“或”组合起来的打印机状态。

- 0x01 设备超时
- 0x08 I/O 出错
- 0x10 已选择
- 0x20 没纸
- 0x40 确认
- 0x80 不忙

当 cmd 等于 0, 且返回值中的设备超时置位时, 表示输出错误。

可移植性 只适用于 IBM PC 及其兼容机

源 程 序

```
#pragma inline
#include <bios.h>
int biosprint(int cmd, int byte, int port)
{
    asm      mov     ah,cmd
    asm      mov     al,byte
    asm      mov     dx,port
    asm      int     17h
    return   AH;
}
```

biostime 返回一天的时间

—biosequ.cas

用 法 long biostime(int cmd,long newtime);原型在 bios.h

说 明 本函数读或设置 BIOS 计时器。该计时器从午夜开始以每秒约 18.2 次滴答的速率计时。

如果 cmd=0, biostime 返回计时器当前值;

如果 cmd=1, 计时器设置为 long 型的 newtime 值。

返 回 值 当 biostime 读 BIOS 计时器(cmd=0)时, 它返回计时器的当前值。

可移植性 只适用于 IBM PC 及其兼容机

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <bios.h>
#pragma warn -rvl
long biostime(int cmd, long newtime)
{
    asm      mov     ah, cmd
    asm      mov     cx, W1(newtime)
    asm      mov     dx, W0(newtime)
```



```
asm    int    lah
asm    mov    ax, dx
asm    mov    dx, cx
}
```

```
#pragma warn .rvl
```

__brk 在近堆栈中修改数据段空间申请。

—brk.cas

用 法 int __brk(void *endds);

原型文件 alloc.h

说 明 将终止值设置为 endds 并相应修改已申请的空间。

返 回 值 成功: 0;

失败: -1, 并将 errno 设置为 ENOMEM(无足够空间)。

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <alloc.h>
#include <errno.h>
int __brk(void *addr)
{
    asm    mov    ax,addr
    asm    mov    dx,sp
    asm    sub    dx,MARGIN
    asm    cmp    ax,dx
    asm    jnb    brkerr
    asm    mov    word ptr __brklvl,ax
    return(0);
brkerr:
    errno = ENOMEM;
    return(-1);
}
```

_brk 修改数据段空间申请。

—fbrk.c

用 法 int _brk(void huge *addr);原型文件 alloc.h

说 明 将终止值修改为 addr 将相应改变已申请的空间。

返 回 值 成功: 0;

失败: -1, 并将 error 设置为 ENOMEM(空间不够)。

源 程 序

```
int _brk(void huge *addr)
{
    if ((addr < _heapbase) || (addr > _heaptop) ||
        !normalize((char far *)addr))
        return(-1);
    else
        return(0);
}
```

brk 改变数据段空间分配

—brk.cas

用 法 int brk(void *endds);

相关函数

用 法 char *sbrk(int incr);

原 型 在 alloc.h

说 明 用于动态地改变分配给调用程序数据段的存储量。这种改变是通过重置程序的截断值进行的, 截断值是数据段结尾处以上的第一个位置的地址。分配的存储空间量随着截断值的增加而增加。

sbrk 设置截断值给 endds, 并相应改变所分配的存储空间。

sbrk 把截断值加上 incr 字节, 并相应改变所分配的存储空间。incr 可以为负, 此时所分配的存储空间数将减少。

如果这样的修改导致所分配的存储空间超过可允许范围, 这两个函数都将不对存储空间作任何改变。

返回值 若调用成功, brk 返回 0 而 sbrk 返回老的截断值。

若调用失败, 两函数都返回-1, 并置 errno 为:

ENOMEM 无足够存储空间

可移植性 适用于 UNIX 系统

参 阅 coreleft

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <alloc.h>
#include <errno.h>
#if (LDATA)
#include <fheap.h>
int brk(void *addr)
{
    return(_brk((void huge *)addr));
}
#else
#include <heap.h>
int __brk(void *addr)
{
    asm      mov     ax,addr
    asm      mov     dx,sp
    asm      sub     dx,MARGIN
    asm      cmp     ax,dx
    asm      jnb     brkerr
    asm      mov     word ptr __brkivl,ax
    return(0);
brkerr:
    errno = ENOMEM;
    return(-1);
}
int brk(void *addr)
{
    return(__brk(addr));
}
#endif
```

bsearch 二分法搜索

—bsearch.c

用 法 void *bsearch(void *key, void *base, int *nelem,
int wdrh, int (*fcmp)());

相关函数 void *lfind(void *key, void *base, int *nelem,
int width, int (*fcmp)());

用 法 void *lsearch(void *base, int *nelem,
int width, int (*fcmp)());

原 型 在 stdlib.h

说 明 bsearch 是用于搜索任意表信息的二分法搜索算法。在调用 bsearch 之前, 表中的各入口项必须按递增顺序排列。

lfind 和 lsearch 也用于搜索表中信息。但由于它们是线性搜索, 所以在调用前无须对表中的各入口项进行排序, 如果由 key 所指的项不在表中, lsearch 将它添加到表

中, 而 `lfind` 不这样做。

- * `base` 指向搜索表的基址 (第 0 个元素)
- * `nelem` 指向包含表中元素个数的整数
- * `width` 包含每一项的字节数
- * `key` 指向待搜索的项

参数 `fcmp` 指向用户所写的比较子程序。该子程序对两项进行比较并返回一比较值

为了搜索表, 这三个搜索函数重复调用地址由 `fcmp` 传递的子程序。

在每次调用比较子程序时, 搜索函数传递两个参数: 一个是指向待搜索项的指针 `key`, 另一个为指向待比较元素的指针 `elem`。

`fcmp` 随时解释搜索关键字和搜索表入口。

返回值 每一个函数都返回与搜索关键字相匹配的第一个表项地址。如果没有匹配项, `bsearch` 和 `lfind` 返回 0。

对于 `bsearch`:

如果搜索关键字	<code>fcmp</code> 返回
<code>> *elem</code>	<code>< 0</code> 的整数
<code>== *elem</code>	0
<code>< *elem</code>	<code>> 0</code> 的整数

对于 `lsearch` 和 `lfind`:

如果搜索关键字	<code>fcmp</code> 返回
<code>!= *elem</code>	<code>!= 0</code> 的整数
<code>== *elem</code>	0

可移植性 适用于 UNIX 系统

参 见 `qsort`

源 程 序

```
#include <stdlib.h> void *bsearch(const void *key, const void *base, register size_t nelem,
size_t width, int cdecl (*fcmp)(const void *, const void *))
{
    char *kmin, *probe;
    int i, j;
    kmin = base;
    while (nelem > 0) {
        i = nelem >> 1;
        probe = kmin + i * width;
        j = (*fcmp)(key, probe);
        if (j == 0)
            return(probe);
        else if (j < 0)
            nelem = i;
        else {
            kmin = probe + width;
            nelem = nelem - i - 1;
        }
    }
    return(0);
}
```

_c0crtinit 开始时被调用的 CRT 初始化过程。

—`crtink.cas`

用 法 `void _c0crtinit(void);`

说 明 被起始代码调用以初始化 VIDEO 结构。只有当用户程序至少调用了一个视屏函数时该模块才被连接。这时通过在所有视屏模块对 `_TurboCRT` 或 `_Video` 的一个外部调用来完成。

源程序

```
void _cbrtinit(void)
{
    _AH = V_GET_MODE;
    _Videolnt();
    _crtinit(AX); /* really only _AL */
    _AH = V_RD_CHAR_ATTR;
    _BH = 0;
    _Videolnt();
    _AH &= 0x7f; /* strip blink bit */
    _video.normattr = _AH;
    _video.attribute = _AH;
}
```

cabs 复数的绝对值

用法 #include <math.h>

double cabs(struct complex znum);

原型在 math.h

说明 见 abs

calloc 分配主存储器

—calloc.c

用法 void *(calloc(unsigned nelem, unsigned elsize);

原型在 stdlib.h, alloc.h

说明 见 malloc

源程序

```
#include <alloc.h> #include <stddef.h>
#include <mem.h>
void *calloc(size_t nelem, size_t elsize)
{
    unsigned long msize;
    register char *cp;
    msize = (unsigned long)nelem * elsize;
    cp = (msize > 0xFFFF) ? NULL : malloc((unsigned)msize);
    if (cp)
        setmem(cp, (unsigned)msize, 0);
    return(cp);
}
```

ceil 上舍入

—ceil.cas

用法 double ceil(double x);

原型在 math.h

说明 见 floor

源程序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#pragma warn -rvl
double ceil (double x)
{
    asm    FLD     DOUBLE (x),
    asm    mov     ax, x [6]
    asm    shl     ax, 1
    asm    cmp     ax, 7FE0h + 06A0h /* 2^53, maximum double precision */
    asm    ja      dlm_beyond
    asm    push    ax /* make a word on the stack */
    asm    mov     bx, sp
```

```

asm    FSTCW  W0 (SS_ [bx])          /* read out the current control word */
asm    mov    ax, 0F3FFh
asm    FWAIT
asm    and    ax, SS_ [bx]            /* mask out the rounding control */
asm    or     ah, 08h                 /* INDP-87 control bits for ceiling mode */
asm    push   ax
asm    FLDCW  W0 (SS_ [bx-2])
asm    pop    ax
asm    FRNDINT                                /* round to integer */
asm    FLDCW  W0 (SS_ [bx])          /* restore original rounding control */
asm    pop    ax
dlim_beyond:                                /* magnitudes beyond 2^53 have no fraction */
dlim_end:    return;
}
#pragma warn .rvl

```

cgets 从控制台读字符串

—cget.cas

用 法 char *cgets(char *string);

原 型 在 conio.h

说 明 见 gets

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <conio.h>
char *cgets(char *s)
{
    #if defined(_LARGE_) || defined(_COMPACT_)
    asm    push    ds                 /* HUGE saves/restores ds itself */
    #endif
    asm    mov     ah, 0Ah             /* cgets (DS:[dx]) */
    asm    LDS     dx, s
    asm    int     21h
    /* find the CR character and replace it with \0 */
    asm    add     dx, 2
    asm    mov     bx, dx
    asm    add     bl, [bx-1]          /* = string length returned from MSDOS */
    asm    adc     bh, 0
    asm    mov     byte ptr[bx], 0 /* zero out the CR */
    #if defined(_LARGE_) || defined(_COMPACT_)
    asm    pop     ds
    #endif
    return(s+2);
}

```

chdir 改变工作目录

—chdir.cas

用 法 int chdir(char *path);

原 型 在 dir.h

说 明 chdir 使得 path 指定的目录变为当前工作目录。path 必须指定一已存在目录。

在 path 参数中可以指定一驱动器号，如：chdir("a:\\turbo C")或 chdir("a:/turboC")

返 回 值 本函数调用成功时，返回 0；否则返回-1，并置 error 为：

ENOENT 路径名或文件名没有找到

可移植性 适用于 UNIX 系统

参 见 mkdir

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <_io.h>

```

```

int chdir(const char *pathP)
{
    pushDS_ asm mov ah, 03Bh
    LDS_ dx, pathP
    int_ 021H
    popDS_
    asm jc chdirFailed
    return(0);
chdirFailed:
    return _IOerror(_AX);
}

```

CheckFile - 构造完整路径名并检查其是否存在。

—searchp.cas

用法 static int pascal near CheckFile(char *pathP, char *driveP,
char *dirP, char *nameP,
char *extP, int mode);

说明 用所给各元件构造完整路径名并检查指定文件的存在性。

返回值 如果指定文件存在, 则返回非负值。

源程序

```

static int pascal near CheckFile(char *pathP, char *driveP, char *dirP,
char *nameP, char *extP, int mode)
{
    register char *bufP;
    struct fblk fbuf;
    bufP = pathP;
    if (*driveP == '\0')
        *driveP = 'A' + getdisk();
    else
        *driveP &= ~0x20;
    *bufP++ = *driveP;
    *bufP++ = ':';
    if (*dirP != '\\') && *dirP != '/' {
        *bufP++ = '\\';
        getcurdir(*driveP - '@', bufP);
        if(*bufP) {
            bufP = CopyUpr(bufP, bufP);
            *bufP++ = '\\';
        }
    }
    bufP = CopyUpr(bufP, dirP);
    if (*(bufP - 1) != '\\') && *(bufP - 1) != '/'
        *bufP++ = '\\';
    bufP = CopyUpr(bufP, nameP);
    if (extP)
        CopyUpr(bufP, extP);
    return (findfirst(pathP, &fbuf, (mode & _PROGRAM) ? 0x27 : 0x37) + 1);
}

```

CheckOpenType 检查文件打开类型。

—fopen.c

用法 static unsigned pascal near
CheckOpenType (const char *type, unsigned *oflagsP,
unsigned *modeP)

原型文件 局限于本模块

说明 该函数返回文件标志值, 如果字符串 type 非法, 则返回 0。

字符串 type 包含的字符的含义如下:

	oflags	mode	f->flags
r	O_RDONLY	不管	_F_READ

w O_CREAT|O_WRONLY|O_TRUNC|O_RDWR F_WRITE

其中, flags 和 mode 是函数 open() 的第二和第三个参数。细节详见 open()。另一字符可选:

..t -正文方式 oflags, f->flags| = O_TEXT,
..b -二进制方式 oflags, f->flags| = O_BINARY, F_BIN

这里要求字符串 type 的第一个字符是 r,w 或 a, 而这以后则可以是任何字符(程序不作检查)。

返回值 见上述说明。

源程序

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <io.h>
static unsigned pascal near
CheckOpenType (register const char *type, unsigned *oflagsP, unsigned *modeP)
{
    extern void      (* exitfopen)();
    extern void      _xclose();
    unsigned         oflags = 0;
    unsigned         mode   = 0;
    unsigned         flags  = 0;
    char             c;
    if ((c = *type++) == 'r')
    {
        oflags = O_RDONLY;
        flags  = _F_READ;
    }
    else if (c == 'w')
    {
        oflags = O_CREAT | O_WRONLY | O_TRUNC;
        mode   = S_IWWRITE;
        flags  = _F_WRITE;
    }
    else if (c == 'a')
    {
        oflags = O_WRONLY | O_CREAT | O_APPEND;
        mode   = S_IWWRITE;
        flags  = _F_WRITE;
    }
    else
        return 0;
    c = *type++;
    if (c == '+' || (*type == '+' && (c == 't' || c == 'b')))
    {
        if (c == '+')
            c = *type;
        /* same modes, but both read and write */
        oflags = (oflags & ~(O_WRONLY | O_RDONLY)) | O_RDWR;
        mode   = S_IREAD | S_IWRITE;
        flags  = _F_READ | _F_WRITE;
    }
    if ('t' == c)
    {
        oflags |= O_TEXT;
    }
    else if ('b' == c)
    {
        oflags |= O_BINARY;
        flags  |= _F_BIN;
    }
}
```

```

    }
    else
    {
        if ((oflags |= ( fmode & (O_TEXT | O_BINARY))) & O_BINARY)
            flags |= _F_BIN;
    }
    exitfopen = xfclose;
    *oflagsP = oflags;
    *modeP = mode;
    return flags;
}

```

_chmod 改变文件的存取方式

—chmoda.cas

用 法 #include <dos.h>

```
int _chmod(char *filename,int func[,int attrib]);
```

原 型 在 io.h

说 明 见 chmod

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <_io.h>
int _chmod (pathname, func, attr) /* chmod has old style definition */
    const char *pathname; /* since prototype has ... */
    int func; int attr; /* for 3rd arg */
{
    pushDS_
asm    mov     ah,43h
asm    mov     al,func
asm    mov     cx,attr
asm    LDS     dx, pathname
asm    int     21h popDS_
asm    jc      _chmodFailed
asm    xchg    ax,cx
asm    return(_AX);
/*
    error detected call __IOerror
*/
_chmodFailed:
    return __IOerror (_AX);
}

```

chmod 改变文件的存取方式

—chmod.c

用 法 #include <sys\stat.h>

```
int chmod(char *filename,int permiss);
```

相关函数

用 法 int _chmod (char *filename,int func[,int attrib]);

原 型 在 io.h

说 明 chmod 根据 permiss 值设置文件 filename 的存取权限, filename 是一指向命名文件的字符串指针。

permiss 可以包含一个或两个符号常量 S_IWRITE 和 S_IREAD(在 sys\stat.h 中定义)。

permiss值	存取权限
S_IWRITE	允许写
S_IREAD	允许读
S_IWRITE S_IREAD	允许读和写

_chmod 函数可以取成设置 MS-DOS 文件属性。如果 func 为 0, 本函数返回文件的当前

MS-DOS 属性; 如果 func 为 1, 则把文件属性置为 attrib 值。

attrib 可以为下列符号常量之一 (在 dos.h 中定义)

FA_RDONLY	只读
FA_HIDDEN	隐含
FA_SYSTEM	系统

返回值 如果改变文件存取模式成功, chmod 返回 0, 否则 chmod 返回-1。

若调用成功, _chmod 返回文件属性字, 否则返回-1。

在出错情况下, errno 被置为下列值之一:

ENOET	文件或路径没有找到
EACCES	不允许此访问

可移植性 chmod 适用于 UNIX 系统

_chmod 只适用于 MS-DOS

参 见 access, open, unlink

源 程 序

```
#include <io.h>
#include <dos.h>
#include <sys\stat.h>
int chmod(const char *filename, int permiss)
{
    register int attrib;
    attrib = _chmod(filename, 0);
    if (attrib == -1)
        return(attrib);
    attrib &= ~FA_RDONLY;
    if ((permiss & S_IWRITE) == 0)    attrib |= FA_RDONLY;
    attrib = _chmod(filename, 1, attrib);
    if (attrib == -1)
        return(attrib);
    return(0);
}
```

chsize 改变文件大小

—chsize.cas

用 法 int chsize <int handle, long size>;

原 型 在 io.h

说 明 本函数改变同 handle 关联的文件的大小。它可以根据 size 的值同文件原来的大小进行比较, 从而截断或扩展文件。

文件的打开模式必须允许写。

如果 chsize 扩展一个文件, 则需要添加空字符(\0); 如果截断文件, 则所有的新的文件结束标志之后的数据将被丢弃。

返回值 若成功, chsize 返回 0; 否则, 返回-1, 并置 errno 为下列值之一:

EACCES	不允许存取
EBADF	非法文件号

可移植性 只适用于 MS-DOS

参 阅 creat, fopen

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <fcntl.h>
#include <io.h>
#define ZEROBUFSIZ 128 /* size of padding buffer */
int chsize (register int fildes, long newSize)
```

```

{
    char    zerobuf[ZEROBUFSIZ];
    long    posn;

asm    mov    ax, 4201h          /* LSeek to current position */
asm    mov    bx, fildes
asm    sub    cx, cx
asm    mov    dx, cx
asm    int    21h
asm    jc     chsizeFailed
asm    mov    W0 (posn), ax
asm    mov    W1 (posn), dx      /* save current position */
asm    mov    ax, 4202h          /* seek to end of file */
asm    sub    cx, cx
asm    mov    dx, cx
asm    int    21h
asm    jc     chsizeFailed
/* See whether we need to extend or truncate. */
asm    cmp    dx, W1 (newSize)
asm    ja     chsizeTruncate
asm    jb     chsizePad
asm    cmp    ax, W0 (newSize)
asm    jae    chsizeTruncate
/* Here we seek to the original end-of-file, and pad the file with zeros. */
chsizePad:
asm    sub    W0 (newSize), ax    /* compute # of bytes to pad */
asm    sbh    W1 (newSize), dx
asm    mov    cx, dx
asm    mov    dx, ax
asm    mov    ax, 4200h          /* seek to old end-of-file */
asm    int    21h
asm    jc     chsizeFailed
/* Fill our temp buffer with (ZEROBUFSIZ) zeros. */
asm    push    ss
asm    pop     es
asm    lea     di, zerobuf
asm    xor     ax, ax
asm    mov     cx, ZEROBUFSIZ / 2
asm    cld
asm    rep     stosw
/* Now write our "zero" buffer repeatedly to the file, until (newSize)
   bytes have been written. */
writeLoop:
asm    mov     cx, ZEROBUFSIZ    /* need more than full buffer ? */
asm    cmp     W1(newSize), 0
asm    jnz     writeChunk
asm    cmp     cx, W0(newSize)
asm    jbe     writeChunk
asm    mov     cx, W0(newSize)
writeChunk:
asm    sub     W0(newSize), cx    /* update byte count */
asm    sbh     W1(newSize), 0
asm    lea     dx, zerobuf
#if LDATA
asm    push    ds
asm    push    ss
asm    pop     ds                /* DS = segment of buffer */
#endif
asm    push    cx
asm    mov     ah, 40h
asm    int     21h              /* write zeros to file */
asm    pop     cx
#if LDATA
asm    pop     ds
#endif
asm    jc     chsizeFailed
asm    cmp     ax, cx            /* make sure everything written */
asm    jne     chsizeFailed
/* See if need to write more. */
asm    mov     ax, W0(newSize)

```

```

asm    or      ax, W1(newSize)
asm    jnz     writeLoop
asm    goto    chsizeSeekback; /* done --> restore file pos */
/* Jump here if an error occurs (AX = DOS error code). */
chsizeFailed:
    return _IOerror (AX);
/* MSDOS truncates a file as a side-effect to function 40 when
   a zero-length write is requested. */
chsizeTruncate:
asm    mov     ax, 4200h /* seek to new end-of-file */
asm    mov     bx, fildes
asm    mov     dx, W0 (newSize)
asm    mov     cx, W1 (newSize)
asm    int     21h
asm    jc      chsizeFailed
asm    mov     ah, 40h /* truncate at desired position */
asm    sub     cx, cx
asm    int     21h
asm    jc      chsizeFailed
chsizeSeekback:
asm    mov     dx, W0 (posn)
asm    mov     cx, W1 (posn)
asm    xchg    si, ax
asm    mov     ax, 4200h /* seek to original place */
asm    int     21h
asm    jc      chsizeFailed
    openfd [fildes] |= O_CHANGED;
    return 0;
}

```

circle 画圆

用 法 #include <graphics.h>

void far circle(int x, int y, int radius);

原 型 在 graphics.h

说 明 见 arc

_clear87 清除浮点状态字

—clear87.cas

用 法 unsigned int _clear87(void);

原 型 在 float.h

说 明 _clear87 清除浮点状态字, 该字是 8087/80287 状态字和由 8087/80287 异常处理程序检测到的其它条件的组合。

返 回 值 返回值的位指出老的浮点状态。对于由 _clear87 返回值完整定义参阅 float.h 文件。

参 阅 _fpreset, _status87

源 程 序

```

#pragma inline
#include <float.h>
#include <dos.h>
unsigned int _clear87 (void)
{
    volatile unsigned Status;
    if ( _8087 ) {
asm    db      0DDh, 07Eh, 0FEh /* FNSTSW Status */
asm    mov     cx, 20
ThisNear:
asm    loop    ThisNear
asm    db      0DBh, 0E2h /* FNCLEX */
    } else {
asm    FSTSW   Status
asm    FCLEX
    }
}

```

```
    return (Status &0x3F);
}
```

cleardevice 清除图形屏幕

用 法 #include <graphics.h>

```
void far clear device(void);
```

原 型 在 graphics.h

说 明 cleardevice 清除整个图形并将 CP(当前位置)移到原点(0,0)。

可移植性 在 Turbo Pascal 5.0 中有相似的程序

参 阅 clearviewport

clearerr 复位错误标志

—clearerr.c

用 法 #include <stdio.h>

```
void clearerr(FILE *stream);
```

原 型 在 stdio.h

说 明 见 ferror

源 程 序

```
#include <stdio.h>
void clearerr (FILE *fp)
{
    fp->flags &= ~(_F_ERR | _F_EOF);
}
```

clearviewport 清除当前视区

用 法 #include <graphics.h>

原 型 在 graphics.h

说 明 本函数清除视区, 并将 CP(当前位置)移到原点(0,0)。

可移植性 在 Turbo Pascal 5.0 中有相似的程序

参 阅 getviewsettings, cleardevice

close 关闭文件句柄

—closea.cas

用 法 int _close(int handle);

原 型 在 io.h

说 明 见 cluse

源 程 序

```
#pragma inline
#include <io.h>
#include < io.h>
int _close (register int fd)
{
    asm    mov     ah, 3eh
    asm    mov     bx, fd
    asm    int     21h
    asm    jc      closeFailed
    _openfd [BX] = -0;
    return 0;
closeFailed:
    return __IOerror (AX);
}
```

close 关闭文件句柄

—close.c

用 法 int close(int handle);

相关函数

用 法 `int _close(int handle);`

原 型 在 `io.h`

说 明 `close` 和 `_close` 都关闭由 `handle` 指出的文件句柄, `handle` 是调用 `_creat`, `creat`, `creatnew`, `creattemp`, `dup`, `dup2`, `_open` 或 `open` 过程中得到的一个文件句柄。

注意: 这些函数并不在文件末尾写一个 `Ctrl_Z` 字符。如果想用 `Ctrl_Z` 结束文件, 必须显式给出该字符。

返 回 值 调用成功时, `close` 和 `_close` 返回 0; 否则返回 -1。

如果 `handle` 是一个不正确的打开文件句柄, 这两个函数都失败, 并置 `errno` 值为:

`EBADF` 非法文件号

可移植性 `close` 适用于 UNIX 系统

`_close` 只适用于 MS-DOS

参 考 `creat`, `dupm`, `fclose`, `fcntl`, `open`

源 程 序

```
#include <io.h>
#include <asmrules.h>
#include <io.h>
#if CPM_cttZ /* see commentary in file "zapcttZ.cas" */ #include <fcntl.h>
#endif
int close (register int handle)
{
    if (handle < 0 || handle >= HANDLE_MAX)
        return IOError (e_badHandle);
    #if CPM_cttZ /* see commentary in file "zapcttZ.cas" */
        if ( !(_openfd[handle] & (O_BINARY | O_DEVICE)) && _openfd[handle] & O_CHANGED)
            AppendCttZ (handle);
    #endif
    _openfd [handle] = ~0;
    return _close(handle);
}
```

closegraph 关闭图形系统

用 法 `#include <graphics.h>`

`void far closegraph(void);`

原 型 在 `graphics.h`

说 明 见 `initgraph`

clock 返回程序执行时钟滴答数

—clock.cas

用 法 `long clock(void);`

原 型 在 `time.h`

说 明 本函数返回程序从开始执行到调用时处理机所化的时钟滴答数, 把该值除以 `CLK_TCK` (在 `time.h` 中定义, 值为 18.2) 就转化为秒值。

参 见 `time`

源 程 序

```
#include <time.h>
#include <asmrules.h>
extern unsigned long _StartTime; /* beginning of program execution */
/* initialized in C0.ASM */
static unsigned char DayFlip; /* keeps track */
#pragma inline
#define l asm
clock_t clock(void)
```

```

{
    clock_t NowTime;
1  xor     ah,ah
1  int     lah                                /* call BIOS for timer ticks */
1  add     BY0(DayFlip), al                  /* record whether day has changed */
1  mov     W0 (NowTime), DX                 /* save return values */
1  mov     W1 (NowTime), CX                 /* which is current time in ticks */
    if(DayFlip)
        NowTime += 0x1800b0L;              /* number of clock ticks in day */
    return(NowTime - _StartTime);           /* return count of ticks */
}

```

clreol 在文本窗口中清除字符到行末

—clreol.c

用 法 void clreol(void);

原 型 在 conio.h

说 明 clreol 在当前文本窗口中将从光标位置到行末的所有字符清除，光标保持不动

可移植性 只适用于 IBM PC 及其兼容机，在 Turbo Pascal 5.0 中有相似的程序。

参 见 clrscr, dcline, window

源 程 序

```

#include < video.h>
#include < conio.h>
void clreol(void)
{
    register int line;
    line = wherey();
    _scroll(UP, _wherex(), line, _video.windowx2, line, 0);
}

```

clrscr 清除文本模式窗口

—clrscr.c

用 法 void clrscr(void);

原 型 在 conio.h

说 明 本函数清除当前窗口，并将光标移到左上角（位置(1,1)）。

可移植性 只适用于 IBM PC 及其兼容机。在 Turbo Pascal 5.0 中有相似的程序

源 程 序

```

#include < video.h>
#include < conio.h>
void clrscr(void)
{
    _scroll(UP, _video.windowx1, _video.windowy1,
    _video.windowx2, _video.windowy2, 0);
    _DL = _video.windowx1; /* position to window 1,1 */
    _DH = _video.windowy1;
    _AH = V_SET_CURSOR_POS;
    _BH = 0;
    _VideoInt();
}

```

comtime

—ctime.c

用 法 static struct tm *comtime(long time, int dst);

原型文件 局限于本模块

说 明 转换长整型 time 并填入时间结构 tm 中。

返 回 值 修改后的时间结构。该结构是静态的，每次都被覆盖。

源 程 序

```

#include <io.h>
#include <time.h>
#include <stdio.h>
static const char Days[12] = {
    31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};
static const char *const Weekday[7] = {
    "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};
static const char *const Months[12] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
static struct tm tm;
static struct tm *comtime(long time, int dst)
{
    int hpery;
    int i;
    int cumdays;
    #pragma warn -sig
    tm.tm_sec = time % 60;
    time /= 60; /* Time in minutes */
    tm.tm_min = time % 60;
    time /= 60; /* Time in hours */
    i = time / (1461L * 24L);
    tm.tm_year = 70 + (i < 2);
    cumdays = 1461 * i;
    time %= 1461L * 24L;
    for (;;) {
        hpery = 365 * 24;
        if ((tm.tm_year & 3) == 0)
            hpery += 24;
        if (time < hpery)
            break;
        cumdays += hpery / 24;
        tm.tm_year++;
        time -= hpery;
    }
    if (dst && daylight &&
        isDST( time % 24, time / 24, 0, tm.tm_year-70 )) {
        time++;
        tm.tm_isdst = 1;
    }
    else
        tm.tm_isdst = 0;
    tm.tm_hour = time % 24;
    time /= 24; /* Time in days */
    tm.tm_yday = (int)time;
    cumdays += (int)time + 4;
    tm.tm_wday = cumdays % 7;
    time++;
    if ((tm.tm_year & 3) == 0) {
        if (time > 60)
            time--;
        else if (time == 60) {
            tm.tm_mon = 1;
            tm.tm_mday = 29;
            return(&tm);
        }
    }
    for (tm.tm_mon = 0; Days[tm.tm_mon] < time; tm.tm_mon++)
        time -= Days[tm.tm_mon];
    tm.tm_mday = time;
    #pragma warn -sig
    return(&tm);
}

```

_control87 处理浮点控制字

—ctrl87.das

用 法 unsigned int _control87(unsigned int newvals,
unsigned int mask);

原 型 在 float.h

说 明 本函数用于取得或改变浮点控制字。

浮点控制字是一个 unsigned int，它的每一位指明浮点包中的某种模式，即精度、无穷大和舍入模式。改变这些模式可以屏蔽或开放浮点异常处理。

_control87 把 mask 中的每一位同 newvals 的对应位相匹配。如果 mask 中的一位等于 1，则 newvals 中的对应位就是包含着浮点控制字中同一位的新值，_control87 便将控制字中的那一位变成该新值。

以下是该工作过程的简单示意：

原控制字:0100 0011 0110 0011

mask:1000 0001 0100 1111

newvals:1110 1001 0000 0101

改变位:1___ __1_0_ 0101

如果 mask=0，则 _control87 返回不改变的浮点控制字。

返 回 值 返回值的位状态反映了浮点控制字的状态。有关 _control87 返回值位状态的完整定义，见 float.h

参 见 _clear87, _fpreset, _status87

源 程 序

```
#pragma inline
#include <float.h>
unsigned int _control87(unsigned int new, unsigned int mask)
{
    volatile unsigned Control;
    volatile unsigned nControl;
asm    fstcw    Control
    nControl = (((Control & ~mask) | (new & mask))) & ~0x0040;
asm    fldcw    nControl
    return(Control & ~0x0040);
}
```

CopyIt 将一个字符串复制到另一个字符串。

—fnsplit.c

用 法 void pascal near CopyIt(char *dst, const char *src,
unsigned maxlen)

原型文件 局限于本模块

说 明 将字符串 SRC 复制到另一个字符串 DST。

返 回 值 无。

源 程 序

```
#include <dir.h>
#include <string.h>
static void pascal near CopyIt(char *dst, const char *src, unsigned maxlen)
{
    if (dst) {
        if(strlen(src) >= maxlen)
        {
            strncpy(dst, src, maxlen);
            dst[maxlen] = 0;
        }
        else
            strcpy(dst, src);
    }
}
```


CopyUpr 复制字符串并转换成大写字母 -- searchp.cas

用 法 void * pascal near CopyUpr(char *dst, char *src)

说 明 将一字符串复制到另一字符串, 并将所有的小写字母转换为大写。

返 回 值 返回 dst 中终结的空字节的地址。

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <dir.h>
#include <string.h>
#include <stdlib.h>
static char PathFile[MAXPATH];
static char drive[MAXDRIVE+1];
static char dir[MAXDIR+1];
static char fname[MAXFILE+1];
static char ext[MAXEXT+1];
static void * pascal near CopyUpr(char *dst, char *src)
{
    pushDS_
asm    LDS     si, src
asm    LES     di, dst
asm    cld
Copying:
asm    lodsb
asm    cmp     al, 'a'
asm    jb      Converted
asm    cmp     al, 'z'
asm    ja      Converted
asm    sub     al, 'a' - 'A'
Converted:
asm    stosb
asm    or      al, al
asm    jnz     Copying
asm    dec     di
asm    popDS_
#pragma warn -sus
    return (void _es *) _DI;
#pragma warn .sus
}
```

coreleft 返回未使用存储器的大小 -- coreleft.cas

用 法 unsigned coreleft(void); (对于微型、小型和中型模式)

 unsigned long coreleft(void); (对于紧缩、大型和巨型模式)

原 型 在 alloc.h

说 明 见 malloc

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <alloc.h>
#if (LDATA)
#include <fheap.h>
unsigned long coreleft(void)
{
    return(farcoreleft());
}
#else
#include <heap.h>
unsigned coreleft(void)
{
asm    mov     ax, sp
asm    sub     ax, word ptr _brklvl
}
```

```

    return(_AX - MARGIN - 40);
}
#endif

```

cos 三角函数

-- cos.cas

用 法 double cos(double x);

原 型 在 math.h

说 明 见 trig

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#include <errno.h>
#include <stddef.h>
static unsigned short NANTRIG [4] = {0,0,0x0420, 0x7FF0};
#pragma warn -rvl
double cos (double x)
{
    asm    FLD    DOUBLE (x)
    asm    mov    ax, 7FF0h
    asm    and    ax, W0 (x [6])          /* extract the exponent field */
    asm    cmp    ax, (53 * 16) + 3FF0h   /* biased version of exponent 53 */
    asm    jae    cos tooLarge
    if ( _8087 >= 3)
    {
        asm    db    OPCODE_FCOS
    }
    else
    {
        asm    FAST_ (_FCOS_)
    }
cos_end:
    return;
cos tooLarge:
    asm    FSTP    ST(0)                  /* pop x from stack */
    #pragma warn -ret
    return _matherr (TLOSS, "cos", &x, NULL, *(double *) NANTRIG);
    #pragma warn .ret
}
#pragma warn .rvl

```

cosh 双曲函数

-- cosh.cas

用 法 double cosh(double x);

原 型 在 math.h

说 明 见 hyperb

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#include <errno.h>
#include <stddef.h>
#pragma warn -rvl
double cosh (double x)
{
    asm    FLD1
    asm    mov    ax, 7FFFh
    asm    FCHS
    asm    and    ax, x [6]              /* TOS = -1.0 */
    asm    FLD    DOUBLE (x)           /* select exponent and most signif. bits */
}

```

```

asm    cmp     ax, 4086h
asm    jnb     cosh_tooBig      /* exp (+-710.475) considered too large */
asm    cmp     ax, 3F20h
asm    jnb     cosh_tiny
cosh_justFits:
asm    FAST_   (_FEXP_)
asm    FLD1
asm    FDIV    st, st(1)        /* Exp (-x) */ asm    FADDP    st(1), st
asm    FSCALE  /* cosh (x) = (exp(x) + exp(-x)) / 2 */
asm    FSTP    st(1)
cosh_end:
    return;
cosh_tooBig:
asm    ja      cosh_over
asm    cmp     WORD [x+4], 033CEh
asm    jnb     cosh_justFits
cosh_over:
asm    FCOMPP   /* discard ST and ST(1) */
#pragma warn -ret
    return_mather (OVERFLOW, "cosh", &x, NULL, HUGE_VAL);
#pragma warn .ret
/*
    cosh is more accurately calculated by the polynomial
    (1 + x^2/2)
    when x is tiny (|x| < 2^-13).
*/
cosh_tiny:
asm    FMUL    ST(0), ST(0)
asm    FSCALE  /* divide by 2 */
asm    FSUBRP  ST(1), ST(0)    /* +1 == - (-1) */
asm    jmp     short cosh_end
}
#pragma warn .rvl

```

country 返回与国家有关的信息

--country.cas

用 法 #include <dos.h>

```

struct country *country(int countrycode,
struct country *countryp);

```

说 明 本函数指定一些与国家有关的信息，如日期、时间和货币，由此函数设置成的值依赖于所用的DOS版本。

```

char co_currstyle; /* currency style */
char co_digits;    /* #of signif,digits in currency */
char co_time;      /* time format */
long co_case;      /* case map */
char co_dasep[2];  /* data separator */
char co_fill[10];  /* filler */
};

```

co_date 中的日期格式如下：

- 0: 表示美国风格的日、月、年日期格式
- 1: 表示欧洲风格的日、月、年日期格式
- 2: 表示日本风格的日、月、年日期格式

由指定的货币显示格式如下：

- 0: 表示货币符号在值之前，且符号和数间无空格
- 1: 表示货币符号在值之后，且符号和数间无空格
- 2: 表示货币符号在值之前，且符号和数间有一空格
- 3: 表示货币符号在值之后，且符号和数间有一空格

返回值 返回指针 countryp

可移植性 只适用于 MS-DOS 3.0 或更高版本

源程序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
struct country *country(int xcode, struct country *cp)
{
    pushDS
    asm LDS dx, cp
    asm mov ah, 38h
    asm mov al, xcode
    asm int 21h
    popDS
    asm jc error
    return(cp);
error:
    return(0);
}
```

cprintf 送格式化输出至屏幕

-- cprintf.c

用法 int cprintf(char *format[, argument, ...]);

原型在 conio.h

说明 见 printf

源程序

```
#include <conio.h>
#ifdef __OLDCONIO__
#pragma inline
#include <asmrules.h>
#include <stdio.h>
#include <_printf.h>
int cdecl cprintf(const char *fmt, ...)
{
    va_list ap;
    va_start (ap, fmt);
    return __vprinter (cputn, stdout, fmt, ap);
}
#else
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <_printf.h>
#include <video.h>
#define BELL '\a'
#define BACKSPACE '\b'
#define LINEFEED '\n'
#define CR '\r'
int cdecl cprintf(const char *fmt, ...)
{
    return __vprinter (__cputn, 0, fmt, (va_list) _va_ptr);
}
#endif
```

cputs 写一字符串到屏幕

--cputs.c

用法 void cputs(char *string);

原型在 conio.h

说明 见 puts

源程序

```

#include <asmrules.h>
#include <conio.h>
#include <string.h>
#ifdef _OLDCONIO_
#pragma inline
int cputs(const char * s)
{
    register int len;
    len = strlen(s);
    #if defined(_COMPACT_) || defined(_LARGE_)
        asm      push    ds                /* HUGE saves/restores ds itself */
    #endif
        asm      mov     ah,40h            /* write ( BX, DS:[dx], CX) */
        asm      mov     bx,1              /* std output = console = fildes 1 */
        asm      mov     cx, len
        asm      LDS     dx, s
        asm      int     21h
    #if defined(_COMPACT_) || defined(_LARGE_)
        asm      pop     ds
    #endif
    return s[len];
}
#else
#include <_video.h>
int cputs(const char * s)
{
    return _cputn(s,strlen(s),0);
}
#endif

```

_creat 创建一个新文件或重写一个已存在文件 -- creat.cas

用 法 #include <dos.h>

int _creat (char *filename ,int attrib);

原 型 在 io.h

说 明 见 creat

源 程 序

int _creat (const char *pathP, int attr)

```

{
    return CreateFile((char *)pathP, attr, 0x3C, O_RDWR | O_BINARY);
}

```

creat 创建一个新文件或重写一个已存在的文件 -- creat.cas

用 法 #include <sys/stat.h>

int creat (char *filename,int permiss);

相关函数 int _creat (char *filename,int permiss);

用 法 int creatnew(char *filename,int attrib);

int creattemp(char *filename,int attrib);

原 型 在 io.h

说 明 creat 根据 filename 指向的文件名创建一个新文件或准备重写一个已存在的文件。permiss 只适用于新创建的文件。

如果文件已存在并且写属性被置位，creat 截断文件长度为字节，文件属性不变。若已存在文件具有只读属性，则 creat 调用失败，文件不变。

creat 调用只检查存取模式字 permiss 中的一位，该位为 UNIX 的用户写允许位。

如果用户写允许位是 1，则该文件是可写的；如果该位为 0，则文件是只读文件。所有其他 MS-DOS 的属性位都置为 0。

permis 可以是下列值之一(在 sys\stat.h 中定义):

permis 值	存取权限
S_IWRITE	允许写
S_IREAD	允许读
S_IREAD S_IWRITE	允许读写

注意: 在 DOS 中, 写允许隐含读允许

用 _CREAT 创建的文件, 总是置为全模式变量 fmode (O_TEXT 或 O_BINARY) 指定的传送方式。

为用特定模式创建一个文件, 可以通过对 _fmode 赋值, 也可以使用 O_CREAT 和 O_TRUNC 任选与所希望的传送方式进行“或”操作后调用 open 函数。如以下调用:

```
open("xmp",O_CREAT|O_TRUNC|O_BINARY,S_IREAD)
```

将建立一个名为 xmp 的二进制只读文件。若该文件已存在, 则截断其长度为 0 字节。

_creat 接受一个 MS-DOS 属性字 attrib。在此调用中可以设置任意一个属性位。文件总是用二进制方式打开。如果创建文件成功, 文件指针被置为文件开头。该文件打开后可读可写。

creattemp 同 _creat 相似, 只是当文件已存在时, 调用 creatnew 返回一个错误, 而文件保持不变。

creattemp 与 _creat 相似, 只是 creattemp 中的 filename 是以反斜杠\结束的路径名。从 filename 给出的目录中可选择一唯一的文件名。新创建的文件名存放在提供 filename 的字符串中。filename 必须足够长以容纳结果文件名。在程序结束时, 此文件不自动删除。

_creat, creatnew 和 creattemp 中的 sttrib 参数可以是下列常量之一(在 dos.h 中定义):

FA_RDONLY	只读属性
FA_HIDDEN	隐含文件
FA_SYSTEM	系统文件

返回值 在调用成功时, 这些函数返回新文件的句柄, 为一非负整数; 否则返回-1。

在发生错误时, errno 被置为下列值之一:

ENOENT	路径或文件名没有找到
EMOENT	打开文件太多
EACCES	无此权限

可移植性 creat 适用于 UNIX 系统。_creat 只适用于 MS-DOS。creatnew 和 creattemp 只适用于 MS-DOS 3.0, 而不能在早期的 DOS 版本工作。

参 见 bsearch, close, dup, _fmode (变量), open, read, write

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
#include <sys\stat.h>
#include <fcntl.h>
extern void (*_exitopen)();
extern void _xclose();
extern unsigned notUmask;
int creat(const char *path, register int mode)
{
    register int fd;
    mode &= notUmask;
    fd = dosCreat (path, (mode & S_IWRITE) ? 0 : 1);
    if (fd >= 0)
    {
        _exitopen = _xclose;
```

```

        _openfd [fd] = fmode | O_RDWR | O_CHANGED |
        (((ioctl (fd, 0) & 0x80) ? O_DEVICE : 0);
    }
    return fd;
}

```

CreateFile 创建一个文件。 -- creat.cas

用法 局限于本模块

说明 被 _creat, creattemp 和 creatnew 用于创建文件。

返回值 成功: 新的文件句柄;

失败: -1, 并将 errno 设置为下列值中的一个:

ENOENT 路径或文件名没找到;

EMFILE 打开文件太多;

EACCESS 不允许存取。

源程序

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
#include <fcntl.h>
#undef creatnew /* Remove the macro version of creatnew() */
static int near pascal CreateFile(char *pathP,
                                   int attr,
                                   unsigned char func,
                                   unsigned oflags)
{
    register int handle;
    pushDS
asm    mov     ah,func
asm    mov     cx,attr
asm    LDS     dx,pathP
asm    int     21h
    popDS
asm    jc      CreateFileFailed
    handle = AX;
    _openfd [handle] = oflags;
    return handle;
CreateFileFailed:
    return __IOerror (AX);
}

```

creatnew 创建一个新文件 -- creat.cas

用法 #include <dos.h>

int creatnew (chr *filename,int attrib);

原型在 io.h

说明 见 creat

源程序

```

int creatnew (const char *pathP, int attr)
{
    return CreateFile((char *)pathP, attr, 0x5B, _fmode | O_RDWR);
}

```

creattemp 创建一个新文件或重写一个已存在文件 -- creat.cas

用法 #include <dos.h>

int creattemp (char *filename,int attrib);

原型在 io.h
说明 见 creat
源程序

```
int creattemp (char *pathP, int attr)
{
    return CreateFile(pathP, attr, 0x5A, _fmode | O_RDWR);
}
```

_crtinit 初始化 _Video 变量。 -- crtinit.cas

用法 void _crtinit(byte newmode)

原型文件 _video.h

说明 初始化 _Video 变量。

源程序

```
void _crtinit(byte newmode)
{
    AL = newmode;
    if (AL > C80 && AL != MONO) AL = C80;
    _video.currmode = AL;
    _AH = V GET_MODE;
    _VideoInt();
    if (_AL != _video.currmode) {
        AL = _video.currmode;
        _AH = V SET_MODE;
        _VideoInt();
        _AH = V GET_MODE;
        _VideoInt();
        _video.currmode = AL;
    }
    _video.screenwidth = _AH;
    _video.graphicsmode = (_video.currmode > C80) &&
        (_video.currmode != MONO);
    _video.screenheight = 25;
    _video.snow = (_video.currmode != MONO) &&
        (!system((char far *)0xF00FFEAL,"COMPAQ")) &&
        (!gainstalled());
    _video.displayptr.u.seg = _video.currmode == MONO ? 0xB000 : 0xB800;
    _video.displayptr.u.off = 0;
    _video.windowx1 = _video.windowy1 = 0;
    _video.windowx2 = _video.screenwidth - 1;
    _video.windowy2 = 24;
}
```

cscanf 从控制台执行格式化输入 -- cscanf.c

用法 int cscanf(char *format [,argument,...]);

原型在 conio.h

说明 见 scanf

源程序

```
#include <conio.h>
#include <stdio.h>
#include <scanf.h>
int cdecl cscanf (const char *fmt, ...)
{
    #pragma warn -sus
    return scanner (
        (int (*)(void *)) getch,
        (void (*)(int ch, void *)) ungetch,
        NULL,
        fmt,
        _va_ptr
    );
}
```



```
);
#pragma warn .sus
}
```

ctime 把日期和时间转化为字符串 -- ctime.c

用 法 `char *ctime(long *clock);`

相关函数 `char *asctime(struct tm *tm)`

用 法 `double difftime(time_t time2, time_t time1);`

`struct tm *gmtime(long *clock);`

`struct tm *localtime(long *clock);`

`void tzset(void);`

原 型 在 `time.h` 说 明 `ctime` 把 `clock` 所指的时间（如函数 `time` 的返回值）转换为下列形式的 26 个字符串：

`Mon Nov 21 11:31:54 1983\n\0`

所有的域都有固定长度。

`asctime` 把以结构形式存储的时间转换为同 `ctime` 字符串形式相同的 26 个字符的字符串。

`difftime` 计算从 `time1` 到 `time2` 间的时间，单位为秒。

`localtime` 和 `gmtime` 返回指向包含修正时间结构的指针。`localtime` 用于修正时区和可能的夏令时间，`gmtime` 把时间直接转换为格林威治时间(GMT)。

全局长整型变量 `timezone` 包含 GMT 和地方标准时间之间的差，单位为秒(在 EST，时区为 5*60*60)。全局变量为非零值，当且仅当应提供美国标准夏令时转换。

程序知道 1974 年和 1975 年中夏令时的换算特性。如果需要的话，这些年的表可以扩充。

`tzset` 是为与 UNIX 兼容而提供的，它在实现过程中什么都不做。

`time.h` 包含文件中的结构说明如下：

```
struct tm {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
}
```

这些量给出了 24 小时的时间，天数(1-31)，月份(0-12)，周日（星期天为 0），年-1900，一年中的天数(0-365)和一个非 0 即表示使用夏令时的标志。

返 回 值 `ctime` 和 `asctime` 返回指向包含日期和时间的字符串的指针。该字符串是静态的，在每次调用时都被重写。

`difftime` 返回计算结果的双精度值

`localtime` 和 `gmtime` 返回修正的时间结构，该结构是静态的，每次调用都被重写。

可移植性 所有函数都适用于 UNIX 系统

参 阅 `getdate, time`

源 程 序

`char *ctime(const long *clock)`

```
{
    return(asctime(localtime(clock)));
}
```

ctrlbrk 设置 Ctrl_Break 处理程序

-- ctrlbrk.c

用 法 void ctrlbrk(*fptr)(void);

原 型 在 dos.h

说 明 Ctrlbrk 设置一个新的由 fptr 所指的 control_break 处理函数。中断向量 0x23 被修改以便使用这个给定名函数。

这个命名函数并不被直接调用。ctrlbrk 建立一个 DOS 中断处理程序来调用它。

该函数可以执行任一操作和系统调用。它并不一定要返回，可以通过 longjmp 返回到程序中任一点。

返 回 值 Ctrlbrk 无返回值。当异常终止当前程序时，处理函数返回 0；其他值使得程序恢复执行。

可移植性 只适用于 MS-DOS

参 见 longjmp, setjmp

源 程 序

```
#include <dos.h>
#include <process.h>
static int (*Hfunc)(void);
void ctrlbrk(int (*fptr)())
{
    Hfunc = fptr;
    setvect(0x23, hentry);
}
```

delay 将程序的执行暂停一段时间（毫秒）

-- delay.asm

用 法 void delay (unsigned milliseconds);

原 型 在 dos.h

说 明 调用 delay 之后，暂停执行当前的程序。参数 milliseconds 指明暂停的毫秒数，精确的时间根据不同的操作环境而变化。

可移植性 只适用于 IBM PC 及其兼容机，在 Turbo Pascal 中有相似的程序。

参 见 sleep, sound

源 程 序

```
NAME DELAY
PAGE 60,132
ifdef _SMALL_
endif
ifdef _COMPACT_
LD = 1
endif
ifdef _MEDIUM_
LP = 1
endif
ifdef _LARGE_
LP = 1
LD = 1
endif
ifdef _HUGE_
LP = 1
LD = 1
HUGE = 1
endif
ifndef NoReverse
Reverse = 1 ; Switch for pushing parameters in reverse order
endif
```

```

ifdef    LP
LPROC   =      1                , Switch for Large Code Segment
else
LPROG   =      0
endif
ifdef    LD LDATA =      1                ; Switch for Large Data Segment
else
LDATA   =      0
endif
ifdef    HUGE
LPROG   =      1
LDATA   =      1
endif
Debug   =      0                ; Switch for internal debug level
ifdef    Debug1
Debug   =      1
endif
ifdef    Debug2
Debug   =      2
endif
ifdef    Debug3
Debug   =      3
endif
HEADERMACRO MODNAME
DSEG    MACRO
IFDEF   HUGE
MODNAME& DATA SEGMENT PARA PUBLIC 'DATA'
        ASSUMECS:MODNAME&_TEXT, DS:MODNAME&_TEXT
ELSE
DATA    SEGMENT WORD PUBLIC 'DATA'
IF      LPROC
        ASSUMECS:MODNAME&_TEXT, DS:MODNAME&_TEXT
ELSE
        ASSUMECS:_TEXT, DS:_TEXT
ENDIF
ENDIF
        ENDM                ;; End macro DSEG
ENDDDS  MACRO
IFDEF   HUGE
MODNAME&_DATA ENDS
ELSE
DATA    ENDS
ENDIF
        ASSUMEDS:NOTHING
        ASSUMECS:NOTHING
        ENDM                ;; End macro ENDDS
PSEG    MACRO
IF      LPROC
MODNAME&_TEXT SEGMENT PARA PUBLIC 'CODE'
        ASSUMECS:MODNAME&_TEXT, DS:MODNAME&_TEXT
ELSE
TEXT    SEGMENT WORD PUBLIC 'CODE'
        ASSUMECS:_TEXT, DS:_TEXT
ENDIF
        IFDEF   HUGE
        ASSUMEDS:MODNAME&_DATA
ELSE
        ASSUMEDS:DGROUP
ENDIF
        ENDM                ;; End macro PSEG
ENDPS   MACRO
IF      LPROC
MODNAME&_TEXT ENDS
ELSE
TEXT    ENDS
ENDIF
        ASSUMECS:NOTHING
        ASSUMEDS:NOTHING
        ENDM                ;; End macro ENDPS
IFDEF   HUGE

```

```

DGROUP GROUP _DATA
ENDIF
DSEG
ENDDS
PSEG
ENDPS

        ENDM                ;; End macro HEADER
;*****
; macros for procedure names and calling conventions
;*****
; address stack frame
        IF      LPROC
ARGOFS EQU    6              ; Offset of arguments
        ELSE
ARGOFS EQU    4              ; Offset of arguments
        ENDIF
GPROC MACRO pname           ;; Name of procedure to define
        IF      LPROC        ;; If large memory model
        ifndef Reverse      ;; If Pascal calling conventions
        PUBLIC pname        ;; Do a PUBLIC declaration
pname PROC FAR              ;; Declare that the model as FAR
        else                ;; If C calling conventions
        PUBLIC &pname        ;; Do a PUBLIC declaration
        &pname PROC FAR      ;; Declare proc as far with underscore
pname equ      &pname        ;; Setup equate for calling proc
        endif              ;; End the IF statement
        ELSE                ;; If small memory model
        ifndef Reverse      ;; If
Pascal calling conventions
        PUBLIC pname        ;; Declare that the model as NEAR
pname PROC NEAR             ;; If C calling conventions
        else
        PUBLIC &pname        ;; Declare proc as near with underscore
        &pname PROC NEAR     ;; Setup equate for calling proc
pname equ      &pname        ;; End the IF statement
        endif              ;; End the IF statement
        ifndef pname&stack
localsz =      pname&ret
parms =      size pname&stack - localsz - ARGOFS
        else
localsz =      0
parms =      0
        endif
PURGE GENDP
GENDP MACRO
        ifndef Reverse      ;; Name of procedure to define
        ;; If Pascal calling conventions
pname ENDP                  ;; Statement to declare out
        else                ;; If C calling conventions
        &pname ENDP          ;; Statement to declare out
        endif              ;; End the IF statement
        ENDM                ;; End macro GENDP
        ENDM                ;; End macro GPROC
LPROC MACRO pname, nopublic ;; Name of procedure to define, public ?
ifndef <nopublic>, <>
        PUBLIC pname
endif
pname PROC NEAR             ;; Declare that the model as NEAR
ifndef pname&stack
localsz =      pname&ret
parms =      size pname&stack - localsz - 4
        else
localsz =      0
parms =      0
        endif
PURGE LENDP
LENDP MACRO
pname ENDP                  ;; Statement to declare out
        ENDM                ;; End macro LENDP
        ENDM                ;; End macro LPROC

```

```

; set up stack frame and push registers
GENTRYMACRO
if      localsz GT 3
    sub    sp, localsz
else
    REPT   localsz
    dec    sp
    ENDM
    ;; End macro REPT localsz
endif

    push    bp
    mov     bp, sp
    push    si
    push    di
    push    ds
    push    es
    ENDM
    ;; End macro GENTRY .

; undo pushes in entry and return
GEXIT MACRO count
    pop     es
    pop     ds
    pop     di
    pop     si
    pop     bp
if      localsz GT 3
    add     sp, localsz
else
    REPT   localsz
    inc     sp
    ENDM
    ;; End macro REPT localsz
endif
ifab    <count>
    ret     count
else
ifdef   Reverse
    ret
else
    ret     parms
endif
endif
    ENDM
    ;; End macro GEXIT
ENTRY MACRO name
    public name
name:
    ENDM
HEADER    delay
PSEG
    loopperms    dw    0
    delaystack   struc
        delaysavebp    dw    ?
        delayret        db    ARGOFs dup (?)
        delaytime       dw    ?
    delaystack   ends
GPROC      delay
    GENTRY
    mov     cx, [bp].delaytime
    mov     ax, cs:_loopperms
    or      ax, ax
    jnz     begindelay
    mov     ax, 40h
    mov     es, ax
    mov     bx, es:[6Ch]
    call    calibrate
    sub     bx, es:[6Ch]
    neg     bx
    mov     ax, 55
    mul     bx
    cmp     cx, ax
    jbe     delayx
    sub     cx, ax
    mov     ax, cs:_loopperms
    begindelay    label    near

```

```

        xor     bx, bx
        mov     es, bx
        mov     di, byte ptr es:[bx]
        jcxz    delayx
waitloop label    near
        mov     si, cx
        mov     cx, ax
        even
counting label    near
;         Garbage values in registers = 0000:0000 (hopefully a constant!)
;         cmp and jne just used to get
;         equivalent loop time here as
;         when estimation loop ran
        cmp     di, byte ptr es:[bx]
        jne     bogustest
bogustest label    near
        loop    counting
        mov     cx, si
        loop    waitloop
delayx  label    near
        mov     ax, cs: loopsperms
        GEXIT
GENDP   delay
LPROC   calibrate,      NoPublic
        push    bx
        push    cx
        push    dx
        push    es
        mov     ax, 40h
        mov     es, ax
        mov     bx, 6Ch
        mov     al, byte ptr es:[bx]
        mov     cx, -1
syncloop label    near
        mov     di, byte ptr es:[bx]
        cmp     al, di
        je      syncloop
        even
countloop label    near
        cmp     di, byte ptr es:[bx]
        jne     donecounting
        loop    countloop
donecounting label    near
        neg     cx
        dec     cx
        mov     ax, 55
        xchg    ax, cx
        xor     dx, dx
        div     cx
        mov     cs: loopsperms, ax
finetune label    near
        mov     al, byte ptr es:[bx]
        mov     cx, -1
syncloop1 label    near
        mov     di, byte ptr es:[bx]
        cmp     al, di
        je      syncloop1
        push    bx
        push    dx
        mov     ax, 55
        push    ax
        call    Delay
IF      Reverse
        pop     ax
ENDIF
        pop     dx
        pop     bx
        cmp     di, byte ptr es:[bx]
        je      calibratex

```

```

                                dec      cs: loopperms
                                jmp      short finetune
calibratex                      label    near
                                pop      es
                                pop      dx
                                pop      cx
                                pop      bx
                                ret
LENDP                           calibrate
ENDPS
END

```

delline 在文本窗口中删去一行

-- inline.c

用法 void delline(void);

原型在 conio.h

说明 delline 删除光标所在的那一行，并把以下各行上移一行，它在当前活动文本窗口中操作。

返回值 本函数不返回值

可移植性 只适用于 IBM PC 及其兼容机，在 Turbo Pascal 中有相似的程序。

参见 'dreol, inline, window

源程序

```

#include <_video.h>
#include <_conio.h>
void delline(void)
/* Deletes the current line on the screen */
{
    _scroll(UP, _video.windowx1, wherey(),
            _video.windowx2, _video.windowy2, 1);
} /* delline */

```

detectgraph 通过检测硬件确定使用的图形驱动程序和模式。

用法 #include <graphics.h>

```

void far detectgraph(int far *graphdriver,
                    int far *graphmode);

```

原型在 graphics.h

说明 见 initgraph

difftime 计算两个时刻之间的时间差

-- difftime.c

用法 #include <time.h>

```

double difftime(time_t time2, time_t time1);

```

原型在 time.h

说明 见 ctime

源程序

```

#include <time.h>
double difftime(time_t time2, time_t time1)
{
    return((double)(time2 - time1));
}

```

disable 屏蔽中断

用法 #include <dos.h>

```

void disable(void);

```

相关函数 void enable(void);

用 法 void geninterrupt(int intr_num);

原 型 在 dos.h

说 明 这些宏设计为程序员提供灵活的硬件中断控制。

disable 宏屏蔽中断, 只有从外设来的不可屏蔽中断(NMI), 仍然是允许的。

enable 宏开放中断, 可以允许任何设备中断发生。

geninterrupt 宏为由 intr_num 给定的中断触发一个软件陷阱。

返 回 值 disable 和 enable 没有返回值, geninterrupt 的返回值依赖于被调用的中断。

可移植性 只适用于 8086 结构

参 见 getvect

div 将两个整数相除, 返回商和余数 -- divt.cas

用 法 #include <stdlib.h>

div_t (int number, int denom);

相关函数

用 法 ldiv_t ldiv (long laumer, long ldenom);

原 型 在 stdlib.h

说 明 div 将两个整数相除, 返回 div_t 类型的商和余数。numer 和 denom 分别是除数和被除数类型 div_t 是一种整型数结构, 在 STDLIB.H 中用 typedef 定义如下:

```
typedef struct {
    int quot; /*quotient*/
    int rem; /*remainder*/
} div_t;
```

ldiv 将两个长整数相除, 返回类型的商和余数。lnumer 和 ldenom 分别是除数和被除数。类型 ldiv 是一个长整型结构, 在 STDLIB.H 中用 typedef 定义如下:

```
typedef struct {
    long quot; /* quotient */
    long rem; /* remainder */
} ldiv_t;
```

返 回 值 每个函数均返回成员为 quot (商) 和 rem (余数) 的结构。

可移植性 ANSI C

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <stdlib.h>
#pragma warn -rvl
div_t div(int numer, int denom)
{
    asm    mov    ax, numer
    asm    cwd
    asm    idiv   W0(denom)
}
#pragma warn .rvl
```

Displacement -- fseek.c

用 法 static int pascal near Displacement(FILE *fp);

源 程 序

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



```
static int pascal near Displacement (FILE *fp)
{
    register level;
    int disp;
    register char *P;
    disp = level = fp->level;
    if (fp->flags & _F_BIN) return level;
#pragma warn -ucp
    P = fp->curp;
#pragma warn .ucp
    while (level--) if ('\n' == *P++) disp ++;
    return disp;
}
```

__DOScmd 用 argv 数组构造一个 DOS 命令行。 -- doscmd.c

用 法 char * pascal __DOScmd(char **argv);

原型文件 _process.h

说 明 申请一个缓冲区并将所有参数串逐个填入该缓冲区。该命令行以命令长度开始，以回车符（不计在命令长度内）结束。

返 回 值 如果成功，__DOScmd 返回指向命令行字符串缓冲区的指针；否则，返回 NULL。

源 程 序

```
#include <_process.h>
#include <stdlib.h>
#include <string.h>
char * pascal __DOScmd(char **argv)
{
    register char **argW;
    register unsigned cmdS, Wrk;
    char *bufP;
    /* Compute the command line size including the NULL string at the
       end of the command line. */
    cmdS = 1; /* Command size byte */
    if ((argW = argv) != NULL && *argW++)
        while (*argW && **argW) {
            Wrk = strlen(*argW++) + 1;
            if ((cmdS + Wrk) > 127)
                break;
            cmdS += Wrk;
        }
    cmdS++; /* Ending Carriage Return */
    /* Allocate a buffer, and concatenate all argument strings */
    if ((bufP = malloc(cmdS)) != NULL) {
        if ((*bufP++ = cmdS - 2) != 0) {
            argW = argv + 1;
            while (*argW && **argW) {
                *bufP++ = ' ';
                bufP = strcpy(bufP, *argW++);
            }
            *bufP++ = '\r';
            return bufP - cmdS;
        }
        else
            return NULL;
    }
}
```

dosCreat 建立一个文件。 -- open.cas

用 法 static int pascal near dosCreat (char *pathP,
unsigned attrib);

说 明 用 DOS 功能 0X3C 建立一个文件。

返 回 值 文件句柄。如出错则为 -1。

源 程 序 \

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
#include <fcntl.h>
#include <sys/stat.h>
extern unsigned notUmask;
static int pascal near dosCreat (char *pathP, unsigned attrib)
{
    pushDS_
    asm      mov      cx, attrib
    asm      mov      ah, 3Ch
    asm      LDS_     dx, pathP
    asm      int      21h      /* AX = creat (DS:DX, CX) */
    popDS_
    asm      jc      dosCreatFailed
    return _AX;
dosCreatFailed:
    return _IOerror (_AX);
}

```

DOSenv 准备 Spawn/Exec 环境。 -- dosenv.c

用 法 char * pascal __DOSenv(char **envV,
char *pathP,
void **envSave);

原型文件 _process.h

说 明 申请一个缓冲区并将所有参数串逐个填入该缓冲区。如果 pathP 变量非零，则将其附加到环境之后，并假设其用于 Spawn 或 Exec。

返 回 值 如果成功，DOSenv 返回指向环境缓冲区的指针；否则，返回 NULL。

源 程 序

```

#include <_process.h>
#include <stdlib.h>
#include <string.h>
char * pascal __DOSenv(char **envV, char *pathP, void **envSave)
{
    register char **envW;
    register unsigned envS;
    char *bufP;
    /* Compute the environment size including the NULL string at the
       end of the environment. (Environment size < 32 Kbytes) */
    envS = 1;
    if ((envW = envV) != NULL)
        for (envS = 0; *envW && **envW; envS += strlen(*envW++) + 1)
            ;
    envS++;
    if (pathP)
        envS += 2 + strlen(pathP) + 1;
    if (envS >= 0x2000)
        return (NULL);
    /* Allocate a buffer */
    if ((bufP = malloc(envS + 15)) != NULL) {
        /* The environment MUST be paragraph aligned */
        *envSave = bufP;
        bufP += 15;
        ((*((unsigned *)&(bufP))) &= 0xFFF0);
        /* Concatenate all environment strings */
        if ((envW = envV) != NULL)
            while (*envW && **envW) {
                bufP = strcpy(bufP, *envW++);
                *bufP++ = '\0';
            }
        *bufP++ = '\0';
        /* Append program name to the environment */
    }
}

```

```

        if (pathP) {
            *((short *)bufP)++ = 1;
            bufP = stpcpy(bufP, pathP);
            *bufP++ = '\0';
        }
        return bufP - envS;
    }
    else
        return NULL;
}

```

dosexterr 获取扩展错误信息

-- dosext.cas

用 法 #include <dos.h>

```
int dosexterr(struct DOSERR *dblkp);
```

原 型 在 dos.h

说 明 在一次调用 MS-DOS 失败后, 本函数在由 dblkp 指向的 DOSERR 结构中填充扩展错误信息。
此结构的定义如下:

```

struct DOSERR {
    int exterror; /*Extended error*/
    char class; /*Error class*/
    char action; /*Action */
    char locus; /*Error locus */
}

```

此结构中的值通过 DOS 调用 0x59 得到。exterror 值为 0 表示前一次 MS-DOS 调用未出错。

返 回 值 dosexterr 返回值为 exterror

可移植性 只适用于 MS-DOS 3.x, 在早期的 MS-DOS 版本中不可用。

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <dos.h>
#pragma warn -use
int dosexterr(struct DOSERROR *eblkp)
{
    /* WARNING: SI,DI and DS are destroyed by the DOS */
    register int SI, DI;
    #if !defined(_HUGE_)
    asm push ds
    #endif
    asm mov ah, 059h
    asm int 021h
    asm LDS di, eblkp
    asm mov [di].exterror, ax
    asm mov [di].class, bh
    asm mov [di].action, bl
    asm mov [di].locus, ch
    #if !defined(_HUGE_)
    asm pop ds
    #endif
    return _AX;
}
#pragma warn ,use

```

函 数 名 dosReadOne - 从文件读取一个字符。 -- zapctriz.cas

用 法 static char pascal near dosReadOne (int fildes)

说 明 从文件读取一个字符。

返 回 值 读取到的字符。

源 程 序

```

static char pascal near dosReadOne (int fildes)
{
    char c = 0;
    pushDS
    asm mov bx, fildes
    asm mov cx, 1
    #if LDATA
    asm push SS
    asm pop DS
    #endif
    asm lea dx, c
    asm mov ah, 3Fh
    asm int 21h /* dosRead (fildes, DS:DX, CX) */
    popDS
    return c; /* no error checking */
}

```

函数名 dosSeekFinalChar - 寻找文件的最终字符。 -- zapctrlz.cas

用法 static void pascal near dosSeekFinalChar (int fildes)

说明 寻找文件结束符。

返回值 无。DS:AX 作为文件结束位置的 MSW:LSW。

源程序

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
#define ctZ 26
static void pascal near dosSeekFinalChar (int fildes)
{
    asm mov bx, fildes
    asm mov cx, -1
    asm mov dx, cx
    asm mov ax, 4200h + SEEK_END
    asm int 21h /* DX:AX = lseek (BX, CX:DX, AL) */
    return; /* no error checking */
}

```

__DOSTimeToU 将 DOS 时间转换为 UNIX 时间。 -- dostimu.cas

用法 long pascal __DOSTimeToU (unsigned long timeStamp);

原型文件 _io.h

说明 timeStamp 是一个 PC DOS 文件时间标记风格的、包含日期和时间位域 (bit-field) 的记录。

返回值 UNIX 时间是从 1970 年 1 月 1 日 00:00:00(GMT)开始的秒数。

源程序

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <time.h>
extern unsigned _monthDay[];
#pragma warn -use
#pragma warn -rvl
long pascal __DOSTimeToU (unsigned long timeStamp)
{
    unsigned hour, yday, year;
    asm mov cx, W0 (timeStamp)
    asm mov dx, W1 (timeStamp)
    asm mov si, cx /* save time in an idle register */
    asm mov di, dx /* save date */
    /* DX holds the date in compacted form:
       yyyy mm dd dd dd year (0 = 1980..119), mon (1..12) day (1..31)
       Lets begin by counting the completed days this year. */
}

```

```

asm    mov    bx, 01E0h
asm    and    bx, dx      /* isolate the month */
asm    mov    cl, 5
asm    shr    bx, cl
asm    and    dx, 1Fh     /* BX = month 1..12, DX = day 1..31 */
asm    mov    cx, 0FE00h
asm    and    cx, di      /* isolate the year */
asm    shr    cx, 1
asm    xchg   cl, ch      /* CX is now the year since 1980. */
asm    test   cl, 3       /* is this a leap year ? */
asm    jnz    dtu_notLeapNow
asm    cmp    hl, 2       /* is it later than February ? */
asm    jna    dtu_notLeapNow
asm    inc    dx          /* add a leap day */
dtu_notLeapNow:
asm    shl    bx, 1
asm    mov    bx, monthDay [bx - 2]
asm    add    bx, dx
asm    dec    bx          /* BX = completed days only ! */
asm    mov    yday, bx
/* Next we need to calculate how many days are counted in the completed
   years since 1970, including 1970. */
asm    add    cl, 10      /* year since 1970 */
asm    mov    year, cx
asm    mov    ax, cx      /* how many leap days occurred ? */
asm    inc    ax          /* count is effectively from 1969. */
asm    shr    ax, 1
asm    shr    ax, 1       /* divided by 4 */
asm    add    bx, ax      /* add leap days to this year's days */
/* In 130 years (start 1970 .. end 2099) there will be 47,481 days. */
asm    mov    ax, 365
asm    mul    cx
asm    add    bx, ax      /* 47,481 will fit into 16 bits. */
/* Each day has 86400 seconds. This is conveniently substituted by
   (2 * 43200). */
asm    mov    ax, 43200
asm    mul    bx
asm    shl    ax, 1
asm    rcl    dx, 1       /* DX:AX = years counted as seconds */
/* swap out the days' total, fetch today's time */
asm    xchg   si, ax
asm    mov    bx, dx      /* BX:SI holds years' seconds. */
/* AX = time field, structured as:
   hhhhh mmmmm sssss hours (0..23), mins (0..59), secs/2 (0..29) */
asm    push   ax
asm    mov    cl, 3
asm    shr    ax, cl      /* AH = hours */
asm    mov    byte ptr hour, ah
asm    mov    byte ptr hour+1, 0
asm    dec    cx
asm    shr    al, cl      /* AL = minutes */
asm    mov    cx, 60
asm    xchg   al, cl
asm    mul    ah
asm    add    ax, cx       /* AX = 60*hours + mins = total mins */
asm    mov    cl, 60
asm    mul    cx
asm    pop    cx
asm    and    cx, 1Fh     /* isolate the two-second counts */
asm    shl    cx, 1
asm    add    ax, cx
asm    adc    di, dh      /* DX:AX = seconds in the day */
asm    add    ax, si
asm    adc    dx, bx      /* DX:AX = total seconds */
/* MSDOS time was local time, so we need to change it to GMT. */
asm    add    ax, W0 (timezone)
asm    adc    dx, W1 (timezone)
asm    cmp    W0 (daylight), 0
asm    jz     dtu_done    /* skip if DST disabled */

```

```

asm    push    ax
asm    push    dx
asm    SI =    isDST( hour, yday, 0, year ) ? 3600 : 0;
asm    pop     dx
asm    pop     ax
asm    sub     ax, si          /* adjust for daylight savings */
asm    shb     dx, 0
dtu_done:
    return;          /* DX:AX contains count of seconds since Unix birthday */
)
#pragma warn .use
#pragma warn .rvl

```

dostounix 转换日期和时间为 UNIX 时间格式 -- timecv.c

用 法 #include <dos.h>

```

long dostounix(struct date *dateptr, struct time
               *timeptr);

```

相关函数 void unixtodost(long utime, struct date *dateptr,
struct time *timeptr);

用 法

原 型 在 dos.h

说 明 dostounix 把从 getdate 和 gettime 返回的日期和时间转换为 UNIX 格式的时间。dateptr 指向 _date 结构, timeptr 指向一包含有效 DOS 日期和时间信息的 time 结构。

unixtodost 把 utime 给定的 UNIX 格式时间转换为 DOS 格式, 并将其填入由 dateptr 和 timeptr 所指的 date 和 time 结构中。

返 回 值 dostounix 返回 UNIX 方式的当前时间: 从格林威治时间(GMT) 1970 年 1 月 1 日 00:00:00 到
至今的秒数。unixtodost 不返回值。

可移植性 两个函数都只适用于 MS-DOS

参 见 ctime, getdate, gettime

源 程 序

```

#include <io.h>
#include <dos.h>
#include <time.h>
static char Days[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
long dostounix(struct date *d, struct time *t)
{
    long          x;
    register int   i;
    register int   days;
    int            hours;
    tzset();
    x = 24L * 60L * 60L * 3652L + timezone; /* get timezone info */
                                           /* Convert from 1980 to
                                           1970 base date GMT */
    i = d->da_year - 1980;
    x += (i >> 2) * (1461L * 24L * 60L * 60L);
    x += (i & 3) * (24L * 60L * 60L * 365L);
    if (i & 3)
        x += 24L * 3600L;
    days = 0;
    i = d->da_mon - 1;          /* Add in months */
    while (i > 0)
    {
        i--;
        days += Days[i];
    }
    days += d->da_day - 1;
    if (d->da_mon > 2 && (d->da_year & 3) == 0)
        days++;                /* Currently in leap year */
    hours = t->tm_hour;
    x += days * 24L * 60L * 60L + hours * 60L * 60L;
}

```

```

    hours = days * 24 + t->ti_hour; /* Find hours */
    if (daylight && __isDST( t->ti_hour, days, 0, d->da_year-1970))
        hours--;
    x += hours * 3600L;
    x += 60L * t->ti_min + t->ti_sec;
    return (x);
}

```

dosWriteNone 向文件写零个字节。 -- open.cas

用 法 static void pascal near dosWriteNone (int handle);

说 明 向文件写 0 个字节。

返 回 值 无。

源 程 序

```

static void pascal near dosWriteNone (int handle)
{
    asm                mov        bx, handle

    asm                sub        cx, cx /* zero length causes truncation */
    asm                sub        dx, dx
    asm                mov        ah, 40h
    asm                int        21h /* dosWrite (files, dont-care, 0) */
                                /* no error checking */
    return;
}

```

DotFound 检查特殊的路径名。 -- fnsplit.c

用 法 int pascal near DotFound(char *pB);

原型文件 局限于本模块

说 明 检查特殊的路径名。

源 程 序

```

static int pascal near DotFound(char *pB)
{
    if (*(pB-1) == '.')
        pB--;
    switch (*pB) {
        case ':':
            if (*(pB-2) != '\\')
                break;

        case '/':
        case '\\':
        case '\0':
            return 1;
    }
    return 0;
}

```

drawpoly 画多边形

用 法 #include <graphics.h>

void far drawpoly(int numpoints, int far *polypoints);

相关函数

用 法 void far fillpoly(int numpoints, int far *polypoints);

说 明 drawpoly 用当前的画线类型和颜色画一顶点数为 numpoints 的多边形。

fillpoly 用当前的线型和颜色画一多边形的外廓(象 drawpoly 那样), 然后用当前填充模式和填充颜色填充此多边形。

polypoints 指向一个整数序列(共有 numpoints*2 个整数)。每一整数对给出多边形一个顶点的 x 和 y 坐标。

注意: 为了画一个有 n 个顶点的封闭图形, 必须将 n+1 个坐标点传给 drawpoly, 此处第 n 个坐标与第 0 个坐标相同

返回值 如果在填充多边形时发生了错误, graphresult 返回-6。

可移植性 在 Turbo Pascal 5.0 中有相似的程序。

参 见 getfillsettings, getlinesettings, getbkcolor, graphresult

dup 复制一个文件句柄 -- dup2.cas

用 法 int dup(int handle);

相关函数

用 法 int dup2(int oldhandle, int newhandle);

原型在 io.h

说 明 dup 和 dup2 都返回一新的文件句柄, 该句柄与原文件句柄有以下相同处:

- 相同的打开文件或设备
- 相同的文件指针(改变了其中一个的文件指针就改变了另一个)
- 相同的存取模式(读、写、读/写)

dup2 返回下一个可用的文件句柄, 它返回值为 newhandle 的新文件句柄。如果与 newhandle 相联的文件是打开的, 在调用 dup2 时被关闭。

handle 和 oldhandle 都是在调用 creat, open, dup, dup2 或 fcntl 中得到的。

返回值 若调用成功, dup 返回新文件句柄, 它为一非负整数; 否则返回-1。

若调用成功, dup2 返回 0; 否则返回-1, 在出现错误时, errno 被置为下列值之一:

EMFILE 打开文件太多

EBADF 无效文件号

可移植性 dup 适用于所有 UNIX 系统

dup2 适用于除系统 III 外的一些 UNIX 系统

参 见 close, creat, open, read, write

源 程 序

```
#pragma inline
#include <io.h>
#include <io.h>
extern void (*_exitopen)();
extern void _xclose();
int dup (int handle)
{
    register int fd;
asm    mov     ah,45h
asm    mov     bx,handle
asm    int     21h
asm    jc      dupFailed
    fd = AX;
    _openfd [fd] = _openfd [handle];
    _exitopen = _xclose;
    return (fd);
dupFailed:
    return __IOerror (_AX);
}
```

dup2 复制一个文件句柄 -- dup2.cas

用 法 int dup2(int oldhandle, int newhandle);

原型在 `io.h`

说明 见 `dup`

源程序

```
#pragma inline
#include <io.h>
#include <io.h>
extern void (*_exitopen)();
extern void _xclose();
int dup2 (register int oldhandle, register int newhandle)
{
asm    mov    ah,46h
asm    mov    bx,oldhandle
asm    mov    cx,newhandle
asm    int    21h
asm    jc     dup2Failed
    _openfd [newhandle] = _openfd [oldhandle];
    _exitopen = _xclose;
    return 0;
dup2Failed:
    return _IOError (_AX);
}
```

ecvt 把一个浮点数转换为字符串

用法 `char ecvt(double value, int ndigit, int *decpt, int *sign);`

相关函数 `char *fcvt(double value, int ndigit, int *decpt, int *sign);`

用法 `char *gcvt(double value, int ndigit, char *buf);`

说明 `ecvt` 把 `value` 转换为 `ndigit` 位数字的以空字符终结的字符串，并返回指向该字符串的指针。相对于串开始处的十进制小数点通过 `decpt` 间接存储(`decpt` 为负表示小数点在返回数字串的左边)。最低数字是舍入位。

`fcvt` 同 `ecvt` 相似，只是为了以 Fortran F 格式输出由 `ndigit` 指定的数字位数，改正位被舍入了。

`gcvt` 把 `value` 转换为以空字符终结的 ASCII 字符串，把结果放在 `buf` 中，并返回一指针。如果可能的话，它也试图产生 Fortran F 格式的 `ndigit` 位有效数字串，否则返回 E 格式数字串(打印就绪)，去掉结尾 0。

返回值 `ecvt` 和 `fcvt` 的返回值均为指向静态数据的指针，其内容在每次调用 `ecvt` 或 `fcvt` 时被重写。

`gcvt` 返回由 `buf` 所指的字符串

可移植性 适用于 UNIX

参见 `printf`

源程序

```
#include <stdlib.h>
#include <_printf.h>
static char _cvtBuf [ _XCVTDIG_ + 2];
char *ecvt (double value, int nDig, int *decP, int *signP )
{
    *decP = _xcvt (& value, (nDig > 1) ? nDig : 1, signP, _cvtBuf, F_8byteFloat);
    return _cvtBuf;
}
```

againstalled 检查 EGA 卡。 -- `crinit.cas`

用法 `static int near pascal againstalled(void);`

说明 通过视屏中断检查 EGA 卡。

返回值 如果找到 EGA 卡则返回 TRUE，否则返回 FALSE。

源程序

```
#pragma inline
#include <asmrules.h>
#include <video.h>
#include <dos.h>
#include <conio.h>
VIDEOREC video = {0};
int Cdecl directvideo = 1;
static int near pascal egainstalled(void)
{
    _AX = 0x1130;
    _BH = 0;
    _DL = 0xff;
    _VideoInt();
    return (unsigned char) (_DL+1);
}
```

ellipse 画一椭圆

用法 `#include <graphics.h>`
`void far ellipse(int x, int y, int stangle, int envangle,`
`int xradius, int yradius);`

原型在 `graphics.h`

说明 见arc

__emit__ 把文字量直接插入到源程序中

用法 `#include <dos.h>`
`void __emit__(argument,...);`

原型在 `dos.h`

说明 `__emit__` 为一内部函数, 它把文字量直接插入到源程序中, 编译后成为机器代码的一部分。

enable 开放中断

用法 `#include <dos.h>`
`void enable(void);`

原型在 `dos.h`

说明 见disable

eof 检测文件结束

-- eof.cas

用法 `int eof(int *handle);`

原型在 `io.h`

说明 eof检测与handle相关联的文件是否已到了末尾。

返回值 如果当前位置是文件末尾, eof返回1; 否则返回0。返回值为-1表示出错, 此时, errno被置为:

EBADF 无效文件号

参见 `ferror`, `perror`

源程序

```
#pragma inline
#include <io.h>
#include <fcntl.h>
#include <io.h>
int eof(int handle) {
    long endPosn;
```



```

#if ! LDATA
asm    push    DS
asm    pop     ES          /* define ES for stos, movs */
#else
asm    push    DS
#endif
asm    mov     cx, qWidth
asm    LES     di, rightP
asm    LDS     si, leftP
/*
  Assert: qWidth is never zero, see test at entry to qsort().
*/
asm    shr     cx, 1        /* test for an odd number of bytes */
asm    jnc     xch_wordLoop
asm    mov     al, ES_ [di]
asm    movsb                    /* swap bytes, advancing pointers */
asm    mov     [si-1], al
asm    jz      xch_end        /* if CX was originally 1 */
xch_wordLoop:
asm    mov     ax, ES_ [di]
asm    movsw                    /* swap words, advancing pointers */
asm    mov     [si-2], ax
asm    loop    xch_wordLoop
xch_end:
#if LDATA
asm    pop     DS
#endif
return;
}

```

exec... 装入并运行其它程序的函数

```

-- exec.asm
-- execl.c
-- execl.e.c
-- execlp.c
-- execlpe.c
-- execv.c
-- execve.c
-- execvp.c
-- execvpe.c

```

用 法 int execl(char *pathname, char *arg0, arg1,..., argn, NULL);
 int execlp(char *pathname, char *arg0, char *arg1,..., argn,
 NULL, char *envp[]);
 int execlpe(char *pathname, char *arg0, arg1,..., argn,
 NULL, char *envp[]);
 int execlp(char *pathname, char *arg0, arg1,..., NULL);
 int execlpe(char *pathname, char *arg0, arg1,..., argn,
 NULL, char *envp[]);
 int execv(char *pathname, char *argv[]);
 int execve(char *pathname, char *argv[], char *envp[]);
 int execvp(char *pathname, char *argv[]);
 int execvpe(char *pathname, char *argv[], char *envp[]);

原 型 在 process.h

说 明 exec...函数族加载并运行(执行)称为子进程的其它程序。当exec...调用成功时,子进程便覆盖父进程。必须有足够的存储空间用于加载和执行子进程。
 pathname是所有调用子进程的文件名,exec...函数使用标准的MS-DOS搜索算法来搜索pathname。

若文件没有扩展名或句点, 将搜索给定的文件名, 如果没有找到, 在文件名后加上.EXE后扩展后再进行搜索过程

若带扩展名, 则按实际的文件名搜索。

若文件名后带句点, 则只搜索无扩展名的文件名。

加在exec...函数族后的后缀p, l, v, e表示指定的函数具有某种功能:

p 表示函数还将在由DOS环境变量PATH的目录中搜索子程序名; 在没有p后缀情况下, 只在根目录和当前目录下搜索。

l 表示参数指针(arg0, arg1, ..., argn)按独立参数传送。一般说来, 当预先知道要传送参数的个数时, 用后缀l。

v 表示参数指针(argv[0], ..., argv[n])按指针数组传送。一般说来, 当要传送的参数可变时, 用后缀v。

e 表示参数envp可以传送到子进程, 这样可以改变子进程的环境。如果没有e后缀, 则子进程继承父进程的环境。

exec...函数族中的每个函数都必须有两个参数指定后缀(l或v)中的一个, 而路径搜索和环境继承后缀(p和e)是任选的。

例如:

- **execl**是一个exec...函数, 它采用独立参数形式, 只在根目录和当前目录下搜索子程序名, 并把父进程的环境传递给子进程。
- **execvpe**是一个exec...函数, 它采用参数指针数组的形式, 把PATH合在对子程序的搜索路径中, 并接受envp参数以改变子进程的环境。

exec...函数必须至少传送一个参数(arg0或argv[0]): 按约定, 该参数为pathname的一个拷贝。对第0个参数使用不同的值不会产生错误。

在MS-DOS 3.x中pathname在子进程中可用; 在早期的DOS版本中, 子程序不能使用所传送的第0个参数(arg0或argv[0])的值。

当使用l后缀时, arg0通常指向pathname, 而arg1...argn指向组成新参数表的字符串。在argn后加上标志NULL, 表示表结束。

当使用e后缀时, 通过参数envp传送一个新的环境设置表。该环境是char类型的数组, 数组中的每一个元素指向一个以空字符终结的字符串, 形式如下:

envvar = value

其中envvar是环境变量名, value是envvar所设置的字符串值。envp[0]中的最后一个元素是NULL。当envp[0]为NULL时, 表示子进程继承父进程的环境设置。

arg0+arg1+...+argn(或argv[0]+argv[1]+...argv[n])的组合总长度(包括空格和参数分隔符)必须小于128字节。终结的NULL字符不算。

当一个exec...函数被调用时, 原已打开的文件在子进程中仍然打开。

返回值 如果调用成功, exe...函数不返回值; 在出错时, exec...函数返回-1, 并设置errno为下列值之一:

E2BIG	参数表太长
EACCES	无此权限
EMFILE	打开文件太多
ENOENT	路径或文件名没有找到
ENOEXEC	格式错误
ENOMEM	无足够存储空间

参 见 abort, atexit, exit, searchpath, spawn, system
源 程 序

```

NAME EXEC
PAGE 60,132
INCLUDE RULES.ASI
; Segments Definitions
Header@
; External References
ExtProc@ IOERROR, PASCAL
ExtSym@ _psp, WORD, CDECL
ExtSym@ _version, WORD, CDECL
; * Miscellaneous equates */
ExeSignature equ 05A4Dh
; /* Data for the Loader */
LdDesc STRUC
LdErrorMsg db 'Exec failure.',00Dh,00Ah
LdStack db 80H dup(?) ;Loader stack
LdPSP dw ? ;PSP address
LdPathName db 80 dup(?) ;File to be loaded
LdAX dw ? ;Parse file name results
LdExeSignature dw ? ;EXE header buffer
LdLength dw ?
LdNbPages dw ?
LdNbItems dw ?
LdHdrSize dw ?
LdMin dw ?
LdMax dw ?
LdSS dw ?
LdSP dw ?
LdChecksum dw ?
LdIP dw ?
LdCS dw ?
LdLoadAddr dw ? ;Load Overlay interface block
LdRelocFactor dw ? ;Relocation factor to be used
LdDesc ENDS
SUBTTL Loader program
PAGE
CSeg@
LoaderDatan db size LdDesc dup (0)
Loader PROC NEAR ;CX = EnvSeg, DX = End of Memory
; /* Setup segment registers */
cli
xor di, di
mov ax, cs
mov ds, ax
mov es, ax
mov ss, ax
lea sp, [di+LdPSP]
; /* Load The Program */
push cx
push dx
mov ax, 4B03h
lea bx, [di+LdLoadAddr]
lea dx, [di+LdPathName]
int 21h
pop dx
pop cx
jh LoadError
xor di, di
mov ss, [di+LdSS]
mov sp, [di+LdSP]
mov bp, sp
xor ax, ax
push ax ;Return to MSDOS for .COM
mov ax, [di+LdPSP]
mov ds, ax
mov es, ax
mov es:[2h], dx ;Set End of Program into PSP
mov es:[2ch], cx ;Set EnvSeg into PSP
mov ax, cs:[di+LdAX] ;AX = validity of FCBs
jmp dword ptr cs:[LdIP]

```

```

/* Fatal Error if unable to load the program */
LoadError label near
        mov     ah, 040h
        mov     bx, 2
        mov     cx, LdStack - LdErrorMsg
        xor     dx, dx
        int     21h                ;Error message on stderr
        mov     ax, 4C02h
        int     21h                ;exit(2)

Loader ENDP
LoaderSize equ ($ - LoaderDatas + 15) / 16
wLoaderSize equ LoaderSize * 8
LoaderVector dd Loader - LoaderDatas
SUBTTL Loader program
PAGE
IF LPROG
pathP equ 6
ELSE
pathP equ 4
ENDIF
IF LDATA
cmdP equ pathP + 4
envP equ cmdP + 4
ELSE
cmdP equ pathP + 2
envP equ cmdP + 2
ENDIF
FileHandle equ 2
MemSize equ FileHandle + 2
EnvSize equ MemSize + 2
EnvAddr equ EnvSize + 4
OldEnv equ EnvAddr + 2
PubProc@ _exec, CDECL
        push    bp
        mov     bp, sp
        sub     sp, OldEnv
        push    si
        push    ds
        push    di
        push    es

/* Open the file to be loaded */
        mov     ax, 3d00h
        pushDS
        LDS     dx, [bp+pathP]
        int     21h
        popDS
        mov     {bp-FileHandle}, ax
        jnb     CopyCmdLine
        jmp     ExecExit

/* Copy the command line into the PSP */
CopyCmdLine label near
IFDEF _HUGE
        mov     ax, seg _psp@
        mov     ds, ax
ENDIF
        mov     es, _psp@
        mov     LoaderDatas.LdPSP, es
        mov     ax, es:[2ch]
        mov     [bp-OldEnv], ax
        mov     di, 080h
        pushDS
        LDS     si, [bp+cmdP]
        lodsb
        mov     dx, si                ;Save cmdP for parse
        stosb
        xor     cx, cx
        mov     cl, al
        inc     cx
        rep     movsb

```

```

/* Parse the command line for FCBs */
    mov     ax, 2901h
    mov     si, dx
    mov     di, 05ch
    int     21h                ;Build FCB 1 in PSP
    mov     byte ptr LoaderDatas.LdAX, al
    mov     ax, 2901h
    mov     di, 06ch
    int     21h                ;Build FCB 2 in PSP
    mov     byte ptr LoaderDatas.LdAX+1, al
    popDS

/* Get the maximum memory block size */
    mov     ah, 4Ah
    mov     bx, -1
    int     21h
    cmp     byte ptr _version@, 3
    jnb     GetMaxMem
    sub     bx, 280h
GetMaxMem    label    near
    mov     [bp-MemSize], bx
/* Compute environment size */
IF LDATA
    mov     ax, [bp+envP]
    mov     dx, [bp+envP+2]
ELSE
    mov     ax, [bp+envP]
    or      ax, ax
    mov     dx, ds
    jnz     DoWeHaveAnEnv
    cwd
ENDIF
DoWeHaveAnEnv    label    near
    mov     bx, ax
    or      bx, dx
    jnz     WeHaveAnEnv
    xor     ax, ax
    mov     di, ax
    jmp     short SetEnvSize
WeHaveAnEnv    label    near
    mov     es, dx
    mov     di, ax
    push    di
    mov     cx, -1
    xor     ax, ax
    cld
GetEnvSize    label    near
    repnz   scasb
    cmp     es:[di], al
    jne     GetEnvSize
    dec     cx
    add     di, 3
    repnz   scasb                /* EnvSize += PathSize */
    dec     cx
    mov     ax, cx
    neg     ax
    pop     di
SetEnvSize    label    near
    mov     [bp-EnvAddr], di
    mov     [bp-EnvAddr+2], es
    add     ax, 15
    mov     cx, 4
    shr     ax, cl
    mov     [bp-EnvSize], ax
    inc     ax
    sub     [bp-MemSize], ax        ;Reserve space for Env
/* Save the pathname of the file to be loaded */
    LDS     si, [bp+pathP]
    push    cs
    pop     es

```



```

CopyPathName    lea     di, LoaderDatus.LdPathName
                 label   near
                 lodsb
                 stosb
                 or      al, al
                 jnz     CopyPathName
/* Process the file according to its type */
                 mov     bx, [bp-FileHandle]
                 mov     ax, [si-5]
                 or      ah, ''
                 push    cs
                 pop     ds
                 mov     di, offset LoaderDatus
                 cmp     ax, 'c.'
                 jne     EXEFile
                 jmp     COMFile
/* Here comes all Exec Error */
ExecError       label   near
                 push    ax
                 mov     ah, 3eh
                 mov     bx, [bp-FileHandle]
                 int     21h
                 pop     ax
                 jmp     ExecExit
/* This file must be an .EXE file */
EXEFile         label   near
                 mov     ah, 3fh
                 mov     cx, LdLoadAddr - LdExeSignature
                 lea     dx, [di + LdExeSignature]
                 int     21h
                 jb      ExecError
                 cmp     [di + LdExeSignature], ExeSignature
                 mov     ax, 11
                 jne     ExecError
                 mov     ax, [di + LdNbPages]
                 xor     dx, dx
                 mov     di, ah
                 mov     ah, al
                 xor     al, al
                 shl     ax, 1
                 rcl     dx, 1          ;DX:AX = NbPages * 512
                 add     ax, [di + LdLength]
                 adc     dx, 0
                 mov     cx, 4
EXECompute      label   near
                 shr     dx, 1
                 rcr     ax, 1
                 loop    EXECompute
                 inc     ax
                 sub     ax, [di + LdHdrSize],
                 add     ax, [di + LdMin]
                 xchg    bx, ax
                 mov     ax, [di + LdPSP]
                 add     ax, 16
                 add     [di + LdCS], ax
                 add     [di + LdSS], ax
                 xchg    ax, bx
                 jmp     short IsItEnough
/* The file to be loaded is .COM file */
COMFile         label   near
                 mov     ax, 4202h
                 xor     cx, cx
                 xor     dx, dx
                 int     21h
                 mov     cx, 4
COMCompute      label   near
                 shr     dx, 1
                 rcr     ax, 1
                 loop    COMCompute

```

```

inc     ax
xchg    bx, ax
mov     ax, [di+LdPSP]
mov     [di+LdCS], ax
mov     [di+LdIP], 0100h
mov     [di+LdSS], ax
add     ax, 16
xchg    ax, bx
IsItEnough label near ;AX = Pgm requirements, BX = Pgm Load Addr
mov     [di+LdLoadAddr], bx
mov     [di+LdRelocFactor], bx
add     ax, LoaderSize
cmp     ax, [bp-MemSize]
mov     ax, 8
jna     $+5
jmp     ExecError
; /* Close the file before load it */
mov     ah, 3eh
mov     bx, [bp-FileHandle]
int     21h
; /* Takes all the memory above the current program */
mov     ah, 4ah
mov     es, [di+LdPSP]
mov     bx, [bp-MemSize]
int     21h
jna     $+5
jmp     ExecError
; /* Move the Loader before loading the overlay */
add     bx, [di+LdPSP]
mov     dx, bx ;Save end of memory
sub     bx, LoaderSize+1
mov     word ptr LoaderVector+2, bx
mov     es, bx
mov     cx, wLoaderSize
mov     si, offset LoaderDatas
xor     di, di
rep     movsw
; /* Copy the environment */
mov     es, [bp-OldEnv]
mov     cx, [bp-EnvSize]
jcxz    ReadyToOverlay
mov     ah, 48h
mov     bx, cx
int     21h
jb      ExecExit
mov     es, ax
xor     di, di
lds     si, [bp-EnvAddr]
add     cx, cx
add     cx, cx
add     cx, cx
rep     movsw
; /* Jump to Loader and overlay the current program */
ReadyToOverlay label near
mov     cx, es
jmp     LoaderVector
; /* All error conditions set doserrno and errno */
ExecExit label near
pop     es
pop     di
pop     ds
pop     si
push    ax
call    _IOERROR@
mov     sp, bp
pop     bp
ret
EndProc@      exec _CDECL_
CSegEnd@

```

```

        END
#include <process.h>
#include <_process.h>
#include <stddef.h>
int execl(char *pathP, char *arg0,...)
{
    return(_LoadProg(_exec, pathP, &arg0, NULL, 0));
}
int execlp(char *pathP, char *arg0,...)
{
    register char **p;
    /* Find the end of the argument list */
    for (p = &arg0; *p++ != NULL; );
    return(_LoadProg(_exec, pathP, &arg0, (char **)p, 0));
}
int execlpe(char *pathP, char *arg0, ...)
{
    return(_LoadProg(_exec, pathP, &arg0, NULL, 1));
}
int execlpe(char *pathP, char *arg0,...)
{
    register char **p;
    /* Find the end of the argument list */
    for (p = &arg0; *p++ != NULL; );
    return(_LoadProg(_exec, pathP, &arg0, (char **)p, 1));
}
int execlp(char *pathP, char *argv[])
{
    return(_LoadProg(_exec, pathP, argv, NULL, 0));
}
int execlpe(char *pathP, char *argv[], char *envV[])
{
    return(_LoadProg(_exec, pathP, argv, envV, 0));
}
int execlp(char *pathP, char *argv[])
{
    return(_LoadProg(_exec, pathP, argv, NULL, 1));
}
int execlpe(char *pathP, char *argv[], char *envV[])
{
    return(_LoadProg(_exec, pathP, argv, envV, 1));
}

```

_exit 终止程序

用 法 void _exit(int status);

原型在 process.h, stdlib.h

说 明 见exit

exit 终止程序

-- exit.c

用 法 void exit(int status);

相关函数

用 法 void exit(int status);

原型在 process.h, stdlib.h

说 明 exit终止调用进程。在退出以前，所有文件被关闭，缓冲输出(正等待输出)内容被写完，所有已登记的“出口函数”(由atexit记入)被调用。

_exit终止调用进程，但不关闭文件，不清除输出缓存，也不调用出口函数。

在两种情况中，status被用来提供调用进程的出口状态，一般说来，值0表示正常出口，非0值表示有错误发生。

返 回 值 两函数都不返回值

可移植性 适用于UNIX系统

参 见 abort, atexit, exec..., spawn...

源 程 序

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

```
extern int _atexitcnt; /* count of atexit function pointers */
```

```

extern  atexit_t      atexittbl[]; /* array of atexit function pointers */
static void dummy(void)
{
}
void (*_exitbuf)(void) = dummy;
void (*_exitfdopen)(void) = dummy;
void (*_exitopen)(void) = dummy;
void exit(int c)
{
    /* Execute "atexit" functions */
    while (atexitcnt--)
        (*atexittbl[atexitcnt])();
    /* Flush and close files and streams */
    (*_exitbuf)();
    (*_exitfdopen)();
    (*_exitopen)();
    _exit(c);
}

```

exp 指数函数, 返回e

-- exp.cas

用法 double exp(double x);

相关函数 double frexp(double value, int *eprtr);

用法 double ldexp(double value, int exp)

double log(double x);

double log10(double x);

double pow(double x, double y);

double pow10(int p);

double sqrt(double x);

原型在 math.h

说明 exp计算指数函数e。

frexp计算尾数为x(小于1的双精度数), 指数为n(整数)的值value=x*2ⁿ, frexp把n存在由eprtr所指的整型变量中。

ldexp计算双精度值value*2^{exp}

log计算x的自然对数

log10计算以10为底的x的对数

pow计算x^y

pow10计算10^p

sqrt计算√x

返回值 所有函数在调用成功时都返回各自的计算值。

exp返回e

frexp返回x(<1), 其中value=x*2ⁿ

ldexp返回x, 其中x=value*2^{exp}

log返回ln(x)

log10返回log(x)

pow返回p, 其中p=x^y

pow10返回x, 其中x=10^p

sqrt返回q, 其中q=√x

有时候, 传给这些函数的参数会产生结果溢出或不可计算。当正确的值溢出时, exp和pow

返回HUGE_VAL。如果结果值很大, 将置error为:

ERANGE 结果超出范围

以下错误将置error为:

EDOM 域错误

- 传给log或log10的参数x小于或等于0
- 传给pow的参数x小于或等于0而y不是一整数。
- 传给pow的参数x, y均为0
- 传给sqrt的参数小于0。

当这些错误发生时:

- log, log10和pow返回负值HUGE_VAL
- sqrt返回0

可以通过matherr函数修改这些函数的错误处理程序。

可移植性 适用于UNIX系统

参 阅 hyperb, trig, matherr

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#include <errno.h>
#include <stddef.h>
#pragma warn -rvl
double exp(double x)
{ asm      FLD      DOUBLE(x)
asm      mov      ax, 7FFFh
asm      and      ax, x [6]      /* select exponent and most signif. bits */
asm      cmp      ax, 4086h
asm      jnb      exp_tooBig     /* exp (+709) is the limit for double */
exp_justFits:
asm      _FAST_   (_FEXP_)
      return;
exp_tooBig:
asm      mov      ax, 0FFFFh     /* force extreme */
asm      ja      exp_excess
asm      mov      ax, x [4]
exp_excess:
asm      test     BY0(x [7]), 80h
asm      jnz      exp_tooTiny
asm      cmp      ax, 02E42h
asm      jb      exp_justFits
asm      mov      si, OVERFLOW
asm      jmp      short exp_err
exp_tooTiny:
asm      cmp      ax, 0232Bh
asm      jb      exp_justFits
asm      mov      si, UNDERFLOW
exp_err:
asm      FSTP     ST(0)          /* discard ST */
#pragma warn -ret
      return _matherr(_SI, "exp", &x, NULL,
                      (UNDERFLOW == _SI) ? 0.0 : HUGE_VAL);
#pragma warn .ret
}
#pragma warn .rvl
```

fabs 绝对值

-- fabs.cas

用 法 double fabs(double x);

原 型 在 math.h

说 明 见 abs

源 程 序

```
#pragma inline
```

```
#include <asmrules.h>
#include <math.h>
#include <math.h>
double fabs(double x)
{ asm    FWAIT
  asm    and    BY0 (x [7]), 7Fh    /* remove any sign bit */
  return x;
}
```

farcalloc 从远堆栈中申请空间。 -- fcalloc.cas

用 法 void *farcalloc(unsigned long nunits,
unsigned long unitsz);

原型文件 alloc.h

说 明 从远堆栈中为一个包含unit个元素、每个元素unitsz个字节的数组申请空间，并将申请到的所有字节都设置为0。

返 回 值 成功：指向最新申请到的空间块的远指针；
失败：NULL。

源 程 序

```
#pragma warn -sig
void far *farcalloc(unsigned long nunits, unsigned long unitsz)
{ register char far *cp, huge *scp;
  unsigned sval;
  nunits *= unitsz;
  if ((cp = farmalloc(nunits)) != NULL)
    for (scp = cp; nunits; scp += sval, nunits -= sval)
    {
      sval = (nunits > 64000L) ? 64000L : (unsigned) nunits;
      setmem((char far *)scp, sval, 0);
    }
  #if (LDATA)
    lsetmem((char far *)scp, sval, 0);
  #endif
  #endif
  return(cp);
}
#pragma warn .sig
```

farcoreleft 返回远堆中未使用存储区大小 -- fcoreleft.c

用 法 long farcoreleft(void);

原 型 在 alloc.h

说 明 见farmalloc

源 程 序

```
#include <alloc.h>
#include <heap.h>
unsigned long farcoreleft(void)
{ return(_heaptop - _brkval - 8);
}
```

farfree 从远堆中释放一块

用 法 void farfree(void far *block);

原 型 在 alloc.h

说 明 见 farmalloc

源 程 序

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

```

#include <dos.h>
#include "fheap.h"
#ifdef _COMPACT_ || defined(_LARGE_) || defined(_HUGE_)
void cdecl free( p )
void far *p;
{
    farfree( p );
}
#endif
/* adds a block to the free-block queue */
static void cdecl insert_free_block( struct header huge *q )
{
    struct header huge *n;
    if( _rover )
    {
        n = _rover->next_free;
        _rover->next_free = q;
        n->prev_free = q;
        q->next_free = n;
        q->prev_free = _rover;
    }
    else
    {
        _rover = q;
        q->prev_free = q;
        q->next_free = q;
    }
}
/* joins two adjacent free blocks together */
static void cdecl join_free_blocks( struct header huge *a,
                                   struct header huge *b )
{
    struct header huge *n;
    a->size += b->size;
    if( _last == b )
    {
        _last = a;
    }
    else
    {
        n = (struct header huge *)((char huge *)b + b->size);
        n->prev_real = a;
    }
    pull_free_block( b );
}
/* frees the last block in the heap and adjusts the break level */
static void cdecl free_last_block( void )
{
    struct header huge *p;
    if( _first == _last )
    {
        brk( _first );
        _first = _last = NULL;
    }
    else
    {
        p = _last->prev_real;
        if( (p->size & 0x00000001) == 0 )
        {
            pull_free_block( p );
            if( p == _first )
                _first = _last = NULL;
            else
                _last = p->prev_real;
            brk( p );
        }
        else
        {
            brk( _last );
            _last = p;
        }
    }
}
/* frees an interior block within the heap */
static void cdecl free_inner_block( struct header huge *q )
{
    struct header huge *n, huge *p;
    --q->size;
    n = (struct header huge *)
        ((char huge *)q + q->size);
    p = q->prev_real;
    if( (p->size & 0x00000001) == 0 && q != _first )
    {
        p->size += q->size;
    }
}

```

```

    n->prev_real = p;
    q = p;
}
else /* add it to the free queue */
{
    insert_free_block( q );
}
if( (n->size & 0x00000001) == 0 ) /* join with the next block? */
{
    join_free_blocks( q, n );
}
}
/* deallocates a block and returns it to the free-block queue */
void cdecl farfree( q )
void far *q;
{
    if ( q == NULL ) /* ANSI requires this */
        return;
    q = (void far *)
        ((char huge *)q - USED_HEADER_SIZE); /* calc block's start address */
    if( (void huge *)q == _last )
    {
        free_last_block();
    }
    else
    {
        free_inner_block( q );
    }
}

```

faralloc 从远堆中分配存储区

-- *malloc.c*

用 法 void far *faralloc (unsigned long size);

相关函数 void far *farcalloc (unsigned long nunits, unsigned long unitsz);

用 法 long farcoreleft(void);

void farfree(void far *block);

void far *farrealloc(void far *block, unsigned long newsize);

原 型 在 alloc.h

说 明 faralloc从远堆中分块一长为size字节的存储块。

farcalloc从远堆中为包含nunits个元素的数组分配存储区，每一区unitsz字节长。

farcoreleft返回远堆中最高已分配存储区以上的未使用存储量。

farfree释放远堆中以前所分配的块

farrealloc调整已分配块大小为newsize，需要的话，把块中内容复制到新位置。

对于从远堆中分配存储区，要注意：

- 所有可用RAM能被分配
- 大于64K的块能被分配
- 远指针用于存取被分配的块

在紧缩、大型和巨型存储模式中，这些函数与通常的存储分配函数不完全相同，但许多地方是相似的。这些函数以unsigned long型为参数，而通常分配函数以unsigned为参数(见malloc函数)。

微型模式不能使用这些函数，因为它没有通常由远指针产生的段前缀。

在小型和中型存储模式中，由faralloc分配的存储块不能通过通常的free来释放。在这两种存储模式中，两种堆是完全不同的。

返 回 值 faralloc和farcalloc返回指向新分配块的指针。如果对新块已没有足够的空间，返回NULL。

farrealloc返回重新分配块的地址。该地址可能与原块地址不同。如果块不能被重新分配，farrealloc返回NULL。

farcoreleft返回最高已分配块与存储区顶部之间自由空间的总量。

可移植性 只适用于MS-DOS

参 阅 malloc

源 程 序

```
#include <stdio.h>
#include <dos.h>
#include "fheap.h"
struct header huge * first = NULL, huge * last = NULL, huge * rover = NULL;
#if defined( _COMPACT_ ) || defined( _LARGE_ ) || defined( _HUGE_ )
void far *cdecl malloc( nbytes )
unsigned int nbytes;
{ return( farmalloc( (unsigned long)nbytes ) );
}
#endif
/* removes a block from the free-block queue */
void cdecl pull_free_block( q )
struct header huge *q;
{ struct header huge *p;
  if( ( _rover = q->next_free ) == q )
  { _rover = NULL;
  }
  else
  { p = q->prev_free;
    _rover->prev_free = p;
    p->next_free = _rover;
  }
}
/* breaks up a large block into two pieces,
one for allocating and one which remains free
returns a pointer to the allocated block */
static void huge *cdecl allocate_partial_block( struct header huge *q,
                                                unsigned long len )
{ struct header huge *n;
  q->size -= len;
  n = (struct header huge *)((char huge *)q + q->size);
  n->size = len + 1;
  n->prev_real = q;
  if( _last == q )
  { _last = n;
  }
  else
  { q = (struct header huge *)((char huge *)n + len);
    q->prev_real = n;
  }
  return( (void huge *)&n->prev_free );
}
/* attempts to extend the heap by pushing the hkr level up
returns a pointer to the last block if successful,
NULL if not */
static void huge *cdecl extend_heap( unsigned long len )
{ struct header huge *q;
  q = _sbrk( len );
  if( (long)q == -1 ) return( (void huge *)NULL );
  q->prev_real = _last;
  q->size = len + 1;
  _last = q;
  return( (void huge *)&_last->prev_free );
}
/* creates a heap from scratch
returns a pointer to the first block if successful, NULL if not */
static void huge *cdecl create_heap( unsigned long len )
{ struct header huge *q;
  q = _sbrk( len );
  if( (long)q == -1 ) return( (void huge *)NULL );
  _first = q;
  _last = q;
  q->size = len + 1;
  return( (void huge *)&q->prev_free );
}
```

```

}
/* attempts to allocate a given number of contiguous bytes on the heap
returns a pointer to the first byte of user space within the allocated block
or NULL if not enough space was available */
void far *cdecl farmalloc( nbytes )
unsigned long nbytes;
{
    struct header huge *q;
    if( nbytes == 0 ) return( NULL );
    /* add the header size and force a 16-byte boundary */
    nbytes = (nbytes + USED_HEADER_SIZE + 15) & 0xfffffff;
    /* create a memory chain if it doesn't yet exist */
    if( ! first )
    {
        return( (void far *)create_heap( nbytes ) );
    }
    /* search for a free block through the memory chain */
    for( q = _rover; q != 0 )
    {
        do
        {
            /* big enough to break up? */
            if( q->size >= nbytes + FREE_HEADER_SIZE + DELTA_FACTOR )
            {
                return( (void far *)allocate_partial_block( q, nbytes ) );
            }
            /* big enough to allocate? */
            if( q->size >= nbytes )
            {
                pull_free_block( q );
                ++q->size; /* mark it as used */
                return( (void far *)((void huge *)&q->prev_free) );
            }
            q = q->next_free;
        } while( q != _rover );
    }
    /* couldn't find a free block big enough, try to extend the heap */
    return( (void far *)extend_heap( nbytes ) );
}

```

farrealloc 调整远堆中的已分配块

-- frealloc.c

用法 void far *farrealloc(void far *block, unsigned long newsize);

原型在 alloc.h

说明 见farmalloc

源程序

```

#include <dos.h>
#include <mem.h>
#include <stdlib.h>
#include "theap.h"
#if defined( __COMPACT__ ) || defined( __LARGE__ ) || defined( __HUGE__ )
void *cdecl realloc(void far *p, unsigned int size)
{
    return( farrealloc( p, (unsigned long)size ) );
}
#endif
/* reallocates a block */
void far *cdecl farrealloc( void far *q, unsigned long newsize )
{
    unsigned long oldsize;
    struct header huge *oldq;
    void huge *newq;
    /* calc block's start address */
    oldq = (struct header huge *) ((char far *)q - USED_HEADER_SIZE);
    oldsize = oldq->size - 1 - USED_HEADER_SIZE;
    newq = farmalloc( newsize );
    if( newq )
    {
        movedata( FP_SEG( q ), FP_OFF( q ),
                  FP_SEG( newq ), FP_OFF( newq ),
                  oldsize < newsize ? (unsigned int)oldsize : (unsigned int)newsize );
        farfree( q );
    }
    return( (void far *)newq );
}

```

fclose 关闭一个流

-- fclose.c

用 法 #include <stdio.h>

int fclose (FILE *stream);

相关函数 int fcloseall (void);

用 法 int fflush (FILE *stream);

int fflushall(void);

原型在 stdio.h

说 明 fclose 关闭名为 stream 的流。一般说来, 所有与 stream 相联的缓冲区在关闭前都被清除。系统分配的缓冲区在关闭时被释放。由 setbuf 和 setvbuf 设置的缓冲区不被自动释放。

fclose 关闭所有除 stdin 和 stdout 外的打开流

fflush 使与一个打开输出流相联的缓冲区内容写到 stream, 如果 stream 是一个打开的输入流, 则清除该缓冲区内容, stream 仍然打开。

fflushall 清除所有与打开输入流相联的缓冲区, 并把所有和打开输出流相联的缓冲区内容写到各自的文件中。跟在 fflushall 后面的读操作从输入文件中读新数据到缓冲区中。

返 回 值 fclose 和 fflush 在调用成功时返回 0; fclose 返回它所关闭的流的总数。当发生错误时, fclose, fcloseall 和 fflush 都返回 EOF。

fflushall 返回一个表示打开输入和输出流总数的整数。

可移植性 适用于 UNIX 系统

参 阅 close, fopen, setbuf

源 程 序

```
#include <stdio.h>
#include <alloc.h>
#include <io.h>
#include <stdio.h>
int fclose(register FILE *fp)
{
    register res = EOF;
    if (fp->token != (short) fp) return res; /* invalid pointer */
    if (fp->bsize)
    {
        if (fp->level < 0 && fflush (fp)) return res;
        if (fp->flags & _F_BUF) free (fp->buffer);
    }
    if (fp->fd >= 0) res = close (fp->fd);
    fp->flags = 0;
    fp->bsize = 0;
    fp->level = 0;
    fp->fd = -1;
    if (fp->istemp != 0)
    {
        unlink(_mkname(NULL, fp->istemp));
        fp->istemp = 0;
    }
    return res;
}
```

fcloseall 关闭打开流

-- fcloseall.c

用 法 int fcloseall(void);

原型在 stdio.h

说 明 见 fclose

源 程 序

```
#include <stdio.h>
int fcloseall (void)
```

```

{
    register FILE *fp;
    register int Nb;
    int Cpt;
    for (Cpt = 0, Nb = OPEN_MAX-5, fp = _streams+5; --Nb; fp++)
        if (fp->fd >= 0) { fclose(fp);
                            Cpt++;
                        }
    return(Cpt);
}

```

fcvt 把一个浮点数转换为字符串

-- **sfv.c**

用 法 char *fcvt (double value, int ndigit, int *decpt, int *sign);

原型在 stdlib.h

说 明 见 ecvt

源 程 序

```

char *fcvt (double value, int nDig, int *decP, int *signP)
/* Identical to ecvt except for the interpretation of nDig.
"nDig" specifies the number of significant digits following the
decimal point.
If nDig < 1 then nDig = 1.
If nDig > 18 then nDig = 18.
Also, the total of digits both left and right of the decimal point shall
not exceed 18. */
{
    *decP = xcvt (& value, (nDig > 0) ? -nDig : 0, signP, _cvtBuf, F_8byteFloat);
    return _cvtBuf;
}

```

fdopen 把流与一个文件句柄相联

-- **fopen.c**

用 法 #include <stdio.h>

FILE *fdopen(int handle, char *type);

原型在 stdio.h

说 明 见 fopen

源 程 序

```

FILE *fdopen (int handle, char *type)
{
    register FILE *fp;
    if (handle < 0 || (fp = getfp()) == NULL)
        return NULL;
    fp->fd = handle;
    return _openfp (fp, NULL, type);
}

```

feof 检测流上的文件结束标志

用 法 #include <stdio.h>

int feof(FILE *stream);

原型在 stdio.h

说 明 见 ferror

ferror 检测流上的错误

用 法 #include <stdio.h>

int ferror (FILE *stream);

相关函数 void clearerr (FILE *stream);

用 法 int feof (FILE *stream);

原型在 stdio.h

说 明 `error` 是一个用于检测给定流读写错误的宏。如果流错误标志被置位, 它保持不变, 直到调用 `clearerr` 或 `rewind` 或关闭流为止。

`clearerr` 设置流错误标志和文件结束标志为 0。

`feof` 是一个用于检测文件结束标志的宏。一旦该标志被置位, 对文件的读操作返回此标志, 直到调用 `rewind` 或文件被关闭。

返回值 如果检测到指定流 `stream` 上的错误, `error` 返回非零值。

`clearerr` 复位指定名流 `stream` 上的错误标志和文件结束标志, 它没有返回值。

如果在指定名流 `stream` 上的最后一次输入操作中检测到文件结束标志, `feof` 返回非零值。

每一次输入操作都复位文件结束标志。

可移植性 适用于 UNIX 系统。

参 阅 `eof`, `fopen`, `getc`, `gets`, `open`, `putc`, `puts`

_fll 填充预读缓冲区。 -- `getc.c`

用 法 `static int near pascal _fll (FILE *fp)`

说 明 填充预读缓冲区, 假设调用程序在调用前已检查流允许权限。

不过, 这里我们仍然检查非法指针。因为没有什么可察觉的性能开销, 并且可以在造成太大危害前捕捉到错误。

返回值 出错: -1(设置进位);

否则: 0(清除进位)。

源 程 序

```
static int near pascal _fll (FILE *fp)
/*      if (fp->token != (short) fp || fp->flags & _F_ERR) */
/*      return EOF; */
    if (fp->flags & F_TERM)
        FlushOutStreams();
    if ((fp->level = read (fp->fd, (fp->curp = fp->buffer), fp->bsize)) > 0)
    {
        fp->flags &= ~_F_EOF;
        return 0;
    }
    else if (0 == fp->level)
    {
        fp->flags = (fp->flags & ~(_F_IN | _F_OUT)) | _F_EOF;
    }
    else
    {
        fp->level = 0;
        fp->flags |= _F_ERR;
    }
    return -1;
}
```

fflush 清除一个流 -- `fflush.c`

用 法 `#include <stdio.h>`

`int fflush (FILE *stream);`

原型在 `stdio.h`

说 明 见 `fclose`

源 程 序

```
#include <stdio.h>
#include <io.h>
int
fflush(register FILE *fp)
{
    register count;
    if (fp->token != (short) fp)
        return EOF;
    if (fp->level >= 0) {
        if (fp->flags & _F_LBUF || fp->curp == &fp->hold) {
            /* validity check */
            /* no output data in buffer */
        }
    }
}
```

```

        fp->level = 0; /* ensure no unget char */
        if (fp->curp == &fp->hold)
            fp->curp = fp->buffer;
    }
    return 0;
}
count = fp->bsize + 1 + fp->level;
fp->level -= count;
if ( (write (fp->fd, (fp->curp = fp->buffer), count) != count) &&
      ((fp->flags & _F_TERM) == 0) ) {
    fp->flags |= _F_ERR;
    return EOF;
}
return 0;
}

```

_fgetc 从流中获取一个字符。 -- getc.c

用法 int _fgetc(FILE *stream);

原型文件 stdio.h

说明 此函数仅供宏 getc()调用。这样做的唯一目的是在调用 fgetc()之前增量层次指示标记。

返回值 读到的字符 (在不带符号转换为整数之后)。

如果文件结束或有错时, 返回 EOF。

源程序

```

int _fgetc(register FILE *fp)
{
    ++fp->level;
    return(fgetc(fp));
}

```

fgetc 从流中读取字符

-- getc.c

用法 #include <stdio.h>

int fgetc (FILE *stream);

原型在 stdio.h

说明 见 getc

源程序

```

int fgetc (register FILE *fp)
{
    unsigned char c;
getAgain:
    if (--(fp->level) >= 0)
        return ((unsigned char) (++fp->curp) [-1]);
    if (++(fp->level) < 0 || fp->flags & (_F_OUT | _F_ERR))
    {
        fp->flags |= _F_ERR;
        return EOF; /* file is in writing mode */
    }
restartStdin:
    fp->flags |= _F_IN;
    if (fp->bsize != 0) /* is the stream buffered ? */
    {
        if (_fill (fp))
            return EOF;
        goto getAgain;
    }
    else /* An unbuffered stream */
    {
        (! _stdinStarted && ((short) fp == (short) stdin))
            if (! isatty (fp->fd))
                fp->flags &= ~_F_TERM;
        setvbuf (fp, NULL, (fp->flags & _F_TERM) ? _IOLBF : _IOFBF, BUFSIZ);
        goto restartStdin;
    }
    do
    {
        if (fp->flags & _F_TERM)

```

```

        FlushOutputStreams();
        if (1 != read (fp->fd, &c, 1))
        {
            if (1 != eof (fp->fd))
                fp->flags |= _F_ERR;
            else
                fp->flags = (fp->flags & ~(_F_IN | _F_OUT)) | _F_EOF;
            return EOF;
        }
    }
    while ('\r' == c && ! (fp->flags & _F_BIN));
    fp->flags &= ~ _F_EOF;
    return ((unsigned char) c);
}
}

```

fgetc 从流中读取字符

-- getc.c

用 法 int fgetc(void);

原型在 stdio.h

说 明 见 getc

源 程 序

```

int fgetc (void)
{
    return getc (stdin);
}

```

_fgetc 从文件流中读取数据。

-- fread.c

用 法 static unsigned pascal near

_fgetc (register void *ptr, size_t n, FILE *fp);

原型文件 局限于本模块

说 明 从输入流 fp 中读取 n 字节到由 ptr 指向的块中。

返 回 值 成功: 0;

失败: 非零。

源 程 序

```

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
static unsigned pascal near _fgetc (register void *ptr, size_t n, FILE *fp)
{
    register int Byte, Temp;
    while(n)
    {
        n++;
        Temp = min(n, fp->bsize);
        if( (fp->flags & _F_BIN) && fp->bsize
            && (n > fp->bsize) && (fp->level == 0))
        {
            n--;
            Temp = 0;
            while(n >= fp->bsize)
            {
                Temp += fp->bsize;
                n -= fp->bsize;
            }
            Byte = _read(fp->fd, ptr, Temp);
            (char *) ptr += Byte;
            if(Byte != Temp)
            {
                n += (Temp - Byte);
                break;
            }
        }
        else
        {
            while (--n && --Temp && (Byte = getc (fp)) != EOF )
                *((char *) ptr)++ = Byte;
            if(Byte == EOF)
                break;
        }
    }
}

```

```

    }
    return n;
}

```

fgetpos 取得当前文件的句柄 -- fgetpos.c

用 法 `#include <stdio.h>`

```
int fgetpos (FILE *stream, fpos_t *pos);
```

相关函数

用 法 `int fsetpos (FILE *stream, const fpos_t *pos);`

原 型 在 `stdio.h`

说 明 `fgetpos` 把与 `stream` 相联的文件句柄的位置保存在 `pos` 所指的地方。

`fsetpos` 把与 `stream` 关联的文件句柄置于新的位置。这个新位置是前次调用流上的 `fgetpos` 所得的值。`fsetpos` 清除 `stream` 所指文件的文件结束标志，并消除对该文件的所有 `ungetc` 操作。在调用 `fsetpos` 之后，文件的下一操作可以是输入或输出。

类型 `fpos_t` 在 `STDIO.H` 中定义如下：

```
typedef long fpos_t
```

返 回 值 若调用成功，`fgetpos` 和 `fsetpos` 都返回 0；若失败，两函数均返回非零值。

参 阅 `fseek`

源 程 序 `#include <stdio.h>`

```
int fgetpos(FILE *stream, fpos_t *pos)
{
    return ((*pos = ftell(stream)) == -1) ? -1 : 0;
}
```

fgets 从流中读取一字符串 -- fgets.c

用 法 `#include, <stdio.h>`

```
char *fgets (char *string, int n, FILE *stream);
```

原 型 在 `stdio.h`

说 明 见 `gets`

源 程 序

```
#include <stdio.h>
char *fgets (char *s, int n, FILE *fp)
{
    register int c = 0;
    register char *P;
    P = s;
    while ('\n' != c && --n > 0 && (c =getc(fp)) != EOF) *P++ = c;
    if (EOF == c && P == s) return NULL;
    *P = 0;
    return (ferror (fp)) ? NULL : s;
}
```

filelength 取文件长度字节数 -- flength.cas

用 法 `long filelength (int handle);`

原 型 在 `io.h`

说 明 `filelength` 返回与句柄 `handle` 相联的文件长度字节数。

返 回 值 调用成功时，`filelength` 返回一个长整型值，即为文件的字节长度；在发生错误时，返回 -1L，并置 `errno` 值为：

`EBADF` 无效文件号

源 程 序

```
#pragma inline
```



```

#include <io.h>
#include <io.h>
long filelength (int handle)
{
    long    Position;
asm    mov    ax,4201h
asm    mov    bx,handle
asm    xor    cx,cx
asm    xor    dx,dx
asm    int    21h
asm    jc     filelengthFailed
asm    push    dx
asm    push    ax
asm    mov    ax,4202h
asm    xor    cx,cx
asm    xor    dx,dx
asm    int    21h
asm    mov    word ptr Position, ax
asm    mov    word ptr Position+2, dx
asm    pop    dx
asm    pop    cx
asm    jc     filelengthFailed
asm    mov    ax,4200h
asm    int    21h
asm    jc     filelengthFailed
    return (Position);
filelengthFailed:
    return __IOerror (_AX);
}

```

fileno 取得文件句柄

用 法 #include <stdio.h>

int fileno (FILE *stream);

原 型 在 stdio.h

说 明 fileno 是一返回指定 stream 的文件句柄的宏。如果 stream 有多个句柄，fileno 返回该流第一次被打开时所赋的文件句柄。

返 回 值 fileno 返回与 stream 相联的整型文件句柄。

可移植性 适用于 UNIX 系统

fillellipse 画一填充椭圆

用 法 #include <graphics.h>

void far fillellipse (int x, int y, int xradius, int yradius);

原 型 在 graphics.h

说 明 本函数以(x, y)为中心，xradius 和 yradius 为水平和垂直半轴画一填充椭圆。椭圆用当前填充颜色和填充方式填充，用当前颜色画边线。

参 阅 arc, circle, ellipse, pieslice

fillpoly 画并填充一个多边形

用 法 #include <graphics.h>

void far fillpoly (int numpoints, int far *polypoints);

原 型 在 graphics.h

说 明 见 drawpoly

findfirst 搜索磁盘目录

-- findfirs.cas

用 法 #include <dir.h>

```
#include <dos.h>
```

```
int findfirst (char *pathname, struct fblk *fblk, int attrib);
```

相关函数

用法 `int findnext(struct fblk *fblk);`

原型在 `dir.h`

说明 `findfirst` 通过 DOS 系统调用 0x4E 对磁盘目录进行搜索。

`pathname` 是一个任选的磁盘驱动器号要寻找的目录和文件名字符串。文件名中可以包含 DOS 匹配符(*或?)等。如果找到了一个匹配的文件, 结构 `fblk` 将用文件目录信息来填入。

`attrib` 是一个 MS-DOS 文件属性字节, 用于在搜索过程中选择符合条件的文件。`attrib` 可以是下列值之一, 在 `dos.h` 中定义:

<code>FA_RDONLY</code>	只读属性
<code>FA_HIDDEN</code>	隐含文件
<code>FA_SYSTEM</code>	系统文件
<code>FA_LABEL</code>	卷 标
<code>FA_DIREC</code>	目 录
<code>FA_ARCH</code>	档 案

有关这些属性的更详细信息, 请参看《MS-DOS 程序员参考手册》。

`findnext` 用于取得与 `findfirst` 中给定的 `pathname` 相匹配的后续文件。`fblk` 是调用 `findfirst` 时填充的同一块。该块包含了继续搜索的必要信息。每调用一次 `findnext`, 将返回一个文件名, 直到目录中再也没有与 `pathname` 相匹配的文件。

结构 `fblk` 的格式如下:

```
struct fblk {
    char ff_reserved[21];    /* Reserved by DOS */
    char ff_attrib;          /* Attribute found */
    int ff_fsize;            /* File size */
    int ff_date;             /* File date */
    long ff_ftime;           /* File time */
    char ff_name[13];        /* Found File name */
};
```

注意: `findfirst` 和 `findnext` 都设置 MS-DOS 磁盘传输地址(DTA)为 `fblk` 地址。

如果需要 DTA 值, 在每次调用 `findfirst` 和 `findnext` 后, 必须把它保存和恢复(使用 `getdta` 和 `setdta`)。

返回值 在成功找到了与搜索路径 `pathname` 相匹配的文件名后, `findfirst` 和 `findnext` 返回 0; 如果再也没有文件找到或文件名中有错误, 则返回 -1, 并置全局变量 `errno` 为下列值之一:

<code>ENOENT</code>	路径或文件名没有找到
<code>ENMFILE</code>	没有更多的文件

可移植性 只适用于 MS-DOS

源程序

```
#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <io.h>
int findfirst(const char *pathname, struct fblk *fblk, int attrib)
{
    asm pushDS
    asm mov ah, 01Ah
    asm LDS dx, fblk
    asm int 021h /* Set the disk transfer address */
}
```

```

asm    mov     ah, 04Eh
asm    mov     cx, attrib
asm    LDS     dx, pathname
asm    int     021h          /* Find first matching file */
asm    popDS
asm    jc      findfirstFailed
return(0);
findfirstFailed:
return __IOerror(_AX);
}

```

findnext 取得下一个匹配 **findfirst** 模式的文件 -- **findfirs.cas**

用 法 `#include <dir.h>` `int findnext (struct fblk *fblk);`

原 型 在 `dir.h`

说 明 见 **findfirst**

源 程 序

```

int    findnext(struct fblk *fblk)
{
asm    pushDS
asm    mov     ah, 01Ah
asm    LDS     dx, fblk
asm    int     021h          /* Set the disk transfer address */
asm    mov     ah, 04Fh
asm    int     021h          /* Find next matching file */
asm    popDS
asm    jc      findnextFailed
return(0);
findnextFailed:
return __IOerror(_AX);
}

```

floodfill 填充一个有界区域.

用 法 `#include <graphics.h>`
`void far floodfill (int x, int y, int border);`

原 型 在 `graphics.h`

说 明 **floodfill** 在位图形设备上填充一块封闭的区域, (x, y)是待填区域的“起点” (seed point)。用颜色 **border** 围起来的区域将用当前填充模式和填充颜色来填充。如果起点在封闭区域内, 则区域内部被填充; 如果起点在封闭区域外, 则区域外部被填充。如果可能, 应该用 **fillpoly** 代替 **floodfill**, 这样可以保持代码同以后版本的兼容性。

返 回 值 若在填充区域过程中出现错误, **graphresult** 返回 -7。

可移植性 在 Turbo Pascal 中有相似的程序

参 阅 **drawpoly**, **gethkcator**, **getfillsetting**, **getlinesetting**, **graphresult**

floor 下舍入 -- **floor.cas**

用 法 `double floor(double x);`

相关函数

用 法 `double ceil (double x);`

原 型 在 `math.h`

说 明 **floor** 求得不大于 **x** 的最大整数

ceil 求得不小于 **x** 的最小整数

返 回 值 **floor** 和 **ceil** 各自返回所求得的值(为双精度型)。

可移植性 适用于 UNIX 系统。

参 阅 **abs**

源程序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#pragma warn -rvl
double floor (double x)
{asm    FLD    DOUBLE (x)
asm    mov    ax, x [6]
asm    shl    ax, 1
asm    cmp    ax, 7FE0h + 06A0h    /* 2^53, maximum double precision */
asm    ja     dlm_beyond
asm    push   ax                    /* make a word on the stack */
asm    mov    bx, sp
asm    FSTCW  W0 (SS [bx])          /* read out the current control word */
asm    mov    ax, 0F3FFh
asm    FWAIT
asm    and    ax, SS [bx]            /* mask out the rounding control */
asm    or     ah, 04h                /* iNDP-87 control bits for floor mode */
asm    push   ax
asm    FLDCW  W0 (SS [bx-2])
asm    pop    ax
asm    FRNDINT
asm    FLDCW  W0 (SS [bx])          /* restore original rounding control */
asm    pop    ax
dlm_beyond:                          /* magnitudes beyond 2^53 have no fraction */
dlm_end:
    return;
}
#pragma warn .rvl
```

flushall 清除所有缓冲区

-- flushall.c

用法 int flushall (void);

相关函数 stdio.h

说明 见 fclose

源程序

```
#include <stdio.h>
int flushall(void)
{
    register FILE *fp;
    register int Nb;
    int Cpt;
    for (Cpt = 0, Nb = OPEN_MAX, fp = _streams; Nb--; fp++)
        if (fp->flags & F_RDWR)
        {
            fflush(fp);
            Cpt++;
        }
    return(Cpt);
}
```

FlushOutStreams 刷新输出流。

-- getc.c

用法 static void pascal near FlushOutStreams(void);

说明 刷新所有已打开的输出流。

返回值 无。

源程序

```
#include <stdio.h>
#include <io.h>
extern int stdinStarted;
static void pascal near FlushOutStreams(void)
{
    register FILE *fp;
    register int Ndx;
```

```

        for (Ndx = OPEN_MAX, fp = streams; Ndx--; fp++)
            if ((fp->flags & (_F_TERM | _F_OUT)) == (_F_TERM | _F_OUT))
                fflush(fp);
    }

```

fmod 计算 X 对 Y 的模, 即 X/Y 的余数

-- fmod.cas

用 法 double fmod (double x, double y);

相关函数

用 法 double modf (double value, double *iptr);

原 型 在 math.h

说 明 fmod 计算 x 对 y 的模(即余数 f, 其中 f 满足: 对一些整数 i, $x=iy+f$, 且 $0 \leq f < y$)。modf 把双精度值 value 分成两部分: 整数和小数部分。它把整数存在 iptr 中, 返回小数部分。

返 回 值 fmod 返回余数 f, 其中 $x=iy+f$ (如上所述)

modf 返回 value 的小数部分

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#pragma warn -rvl
double fmod (double x, double y) {
asm    FLD    DOUBLE (y)
asm    mov    ax, y [6]
asm    shl    ax, 1
asm    jz     mod_resultZero      /* ignore the sign bit */
asm    cmp    ax, 0FFE0h          /* if the divisor is zero */
asm    jnb    mod_isX             /* if y is infinite */
asm    FLD    DOUBLE (x)
asm    mov    ax, x [6]
asm    shl    ax, 1
asm    jz     mod_xZero           /* if x is zero */
asm    cmp    ax, 0FFE0h
asm    jnb    mod_overflow       /* if x is infinite */
mod_keepTrying:
asm    FPREM
asm    push    bx
asm    mov     bx, sp
asm    FSTSW  W0 (SS_ [bx])      /* C2 will be set if not yet finished */
asm    FWAIT
asm    pop     ax
asm    sahf
asm    jp     mod_keepTrying     /* C2 bit maps onto parity flag. */
asm    FSTP   st(1)             /* discard the divisor */
mod_end:
    return;
/* If the divisor is infinite then return the dividend. */
mod_isX:
asm    FSTP   st(0)              /* pop y off the stack */
asm    FLD    DOUBLE (x)
asm    jmp    short mod_end
/* All other forms of overflow are mapped onto zero. */
mod_xZero:
mod_overflow:
asm    FSTP   st(0)              /* pop x off the stack */
mod_resultZero:
asm    FSTP   st(0)              /* pop y off the stack */
asm    FLDZ
asm    jmp    short mod_end
}
#pragma warn rvl

```

fnmerge 建立新文件名 -- fnmerge.c

用 法 `#include <dir.h>`

```
void fnmerge(char *path, char *drive, char *dir,
             char *name, char *ext);
```

相关函数 `int fnsplit (char *path, char *drive, char *dir,`

用 法 `char *name, char *ext);`

原 型 在 `dir.h`

说 明 **fnmerge** 用其各成份参数建立一文件名。文件的完整路径名为:

`X:\DIR\SUBDIR\NAME.EXT`

其中:

`X` 由 `drive` 给定

`\DIR\SUBDIR\` 由 `dir` 给定

`NAME.EXT` 由 `name` 和 `ext` 给定

fnsplit 把文件名路径 `path` 分成四个成份, 并把这些成份分别存在由 `drive`, `dir`, `name` 和 `ext` 所指的字符串中(每一成份都需要, 但也可为 `NULL`, 表示相应成份被分析但不存储)。

这些字符串的最大长度分别由常量 `MAXDRIVE`, `MAXDIR`, `MAXPATH`, `MAXNAME`, `MAXEXT` 给定(在 `dir.h` 中定义), 每一最大值中都包括空字符终结符。

常量	最大值	字符串
<code>MAXPATH</code>	80	<code>path</code>
<code>MAXDRIVE</code>	3	<code>drive</code> , 包括冒号(:)
<code>MAXDIR</code>	66	<code>dir</code> , 包括开始和结尾反斜杠(\)
<code>MAXFILE</code>	9	<code>name</code>
<code>MAXEXT</code>	5	<code>ext</code> , 包括开始圆点(.)

fnsplit 假设有足够的空间来存储每个非 `NULL` 成份。**fnmerge** 假设有足够的空间来存储所组成的完整路径名, 完整路径名的最大长度为 `MAXPATH`。

当 **fnsplit** 分解 `path` 时, 按以下方法处理标点:

- `drive` 后跟冒号(如 `C:`, `A:` 等)
- `dir` 前和后跟反斜杠(如 `\turbo\include\source\` 等)
- `ext` 前有一圆点(如 `.C`, `.EXE` 等)

这两个函数是互逆的, 如果用 **fnsplit** 把一给定 `path` 分解, 接着把各分解成份用 **fnmerge** 并起来, 又可得到 `path`。

返 回 值 **fnsplit** 返回一整数(由五个标志位组成, 在 `dir.h` 中定义)表示哪些完整路径成份在 `path` 中存在。这些标志和代表的对应成份为:

<code>EXTENSION</code>	扩展名
<code>FILENAME</code>	文件名
<code>DIRECTORY</code>	目录名(可能为子目录)
<code>DRIVE</code>	驱动器号(见 <code>dir.h</code>)
<code>WILDCARDS</code>	匹配符(*或?)

可移植性 只适用于 MS-DOS 系统

源 程 序

```
#include <dir.h>
#include <string.h>
void fnmerge(register char *pathP, const char *driveP, const char *dirP,
const char *nameP, const char *extP)
{
    if (driveP && *driveP)
    {
        *pathP++ = *driveP++;
    }
}
```

```

        *pathP++ = '\0';
    }
    if (dirP && *dirP)
    {
        pathP = strcpy(pathP, dirP);
        if ((*pathP-1) != '\\') && (*pathP-1) != '/' *pathP++ = '\\';
    }
    if (nameP) pathP = strcpy(pathP, nameP);
    if (extP) pathP = strcpy(pathP, extP);
    *pathP = 0;
}

```

fnspplt 把一个完整路径名分解为各组成成份 -- fnspplt.c

用 法 #include <dir.h>

```

int fnspplt (char *path, char *drive, char *dir, char *name,
             char *ext);

```

原 型 在 dir.h

说 明 见 fnmerge

源 程 序

```

int fnspplt(const char *pathP, char *driveP, char *dirP,
            char *nameP, char *extP)
{
    register char *pB;
    register int Wrk;
    int Ret;
    char buf[MAXPATH+2];
    /* Set all string to default value zero */
    Ret = 0;
    if (driveP)
        *driveP = 0;
    if (dirP)
        *dirP = 0;
    if (nameP)
        *nameP = 0;
    if (extP)
        *extP = 0;
    /* Copy filename into template up to MAXPATH characters */
    pB = buf;
    while (*pathP == ' ')
        pathP++;
    if ((Wrk = strlen(pathP)) > MAXPATH)
        Wrk = MAXPATH;
    *pB++ = 0;
    strncpy(pB, pathP, Wrk);
    *(pB + Wrk) = 0;
    /* Split the filename and fill corresponding nonzero pointers */
    Wrk = 0;
    for (; ) {
        switch (*--pB) {
            case '.' :
                if (!Wrk && (*(pB+1) == '\\0'))
                    Wrk = DotFound(pB);
                if ((!Wrk) && ((Ret & EXTENSION) == 0)) {
                    Ret |= EXTENSION;
                    CopyIt(extP, pB, MAXEXT - 1);
                    *pB = 0;
                }
                continue;
            case ':' :
                if (pB != &buf[2])
                    continue;
            case '\\0' :
                if (Wrk) {
                    if (*++pB)
                        Ret |= DIRECTORY;
                    CopyIt(dirP, pB, MAXDIR - 1);
                }
                return Ret;
        }
    }
}

```

```

        *pB-- = 0;
        break;
    }
    case '/':
    case '\\':
        if (!Wrk) {
            Wrk++;
            if (*++pB)
                Ret |= FILENAME;
            CopyIt(nameP, pB, MAXFILE - 1);
            *pB-- = 0;
            if (*pB == 0 || (*pB == '?' && pB == &buf[2]))
                break;
        }
        continue;
    case '*':
    case '?':
        if (!Wrk)
            Ret |= WILDCARDS;
    default:
        continue;
    }
    break;
}
if (*pB == ':') {
    if (buf[1])
        Ret |= DRIVE;
    CopyIt(driveP, &buf[1], MAXDRIVE - 1);
}
return (Ret);
}

```

fopen 打开一个流

-- fopen.c

用 法 #include <stdio.h>

FILE *fopen (char *filename, char *type);

相关函数 FILE *fdopen (int handle, char *type);

用 法 FILE *freopen (char *filename, char *type, FILE *stream);

原 型 在 stdio.h

说 明 fopen 打开由 filename 指定的文件，并使其与流 stream 相联。它返回用于后续操作中指明流 stream 的指针。

fdopen 使流 stream 与一个从 creat, dup, dup2 或 open 得到的文件句柄相联。stream 类型必须与打开文件句柄 handle 的模式相匹配。

freopen 用指定文件名代替打开的流 stream。不管打开是否成功，原流 stream 都被关闭。

freopen 非常有助于改变联在 stdin, stdout 和 stderr 上的文件。

在这些调用中的类型字符串 type 可以是下列值之一：

- r 打开用于只读
 - w 创建用于写
 - a 附加。打开用于写在文件末尾，当文件不存在时，创建用于写
 - r+ 打开一已存在文件用于更新(读和写)
 - w+ 创建一新文件用于写
 - a+ 打开用于附加。打开(如果文件不存在，则创建)文件用于在末尾更新
- 为指明一给定文件用文本方式打开或创建，可以在type后面加上t(如rt。

w+t 等);如果以二进制方式打开或创建,则加上 b(如 wb, a+h 等)。

若 type 中没有给出 b 或 t,则由全局变量 _fmode 决定打开或创建方式。

若 _fmode 为 O_BINARY,则文件以二进制方式打开

若 _fmode 为 O_TEXT,则文件以文本方式打开

常量 O_BINARY 和 O_TEXT 在fcntl.h 中定义

当文件打开用于更新时,在结果流 stream 上既可进行输入也可进行输出。但如果没有给出 fseek 或 rewind 调用,输出不能直接跟在输入后面,同样,如果没有给出 fseek 或 rewind 调用,输入也不能直接跟在输出后面,在遇到文件结束时不能进行输入。

返回值 若调用成功,每一函数都返回各自新打开的流 stream。freopen 返回参数 stream。

在出现错误时,均返回 NULL。

可移植性 适用于 UNIX 系统。fopen 在 Kernighan 和 Ritchie 所著书中定义。

参 见 creat, dup, fclose, ferror, _fmode(变量), fread, fseek, getc, gets, open, putc, puts, rewind, setbuf, setmode

源 程 序

```
FILE *fopen (const char *filename, const char *type)
{
    register FILE *fp;
    if ((fp = getfp()) == NULL)
        return NULL;
    else
        return _openfp (fp, filename, type);
}
```

_fpreset 重新初始化浮点数学包 -- fpreset.c

用 法 void _fpreset (void);

原 型 在 float.h

说 明 _fpreset 重新对浮点数学包进行初始化。本函数通常与 signal, system 或 exec..., spawn... 函数配合使用。

注意: 在 DOS 3.X 之前的版本中,如果在使程序中使用 8087/80287 协处理器,则子进程(由 system 或 exec... 或 spawn... 函数执行)可以改变父进程的浮点状态。

如果使用了 8087/80287,则必须注意以下几点:

- * 在计算一个浮点表达式时,不得调用 system, exec... 或 spawn... 函数
- * 如果子进程有可能用 8087/80287 执行了一次浮点操作,则在调用 system, exec... 或 spawn... 之后,必须调用 _fpreset 复位浮点状态

返回值 本函数不返回值

参 见 exec..., longjmp, signal, spawn..., system

源 程 序

```
#include <float.h>
/* The function _fpreset() is in the emulator library and is */
/* used during the initialisation of the emulator (_emulst). */
/* As all the emulator functions which are model-independent, */
/* _fpreset() is a FAR function. */
void pascal far EMURESET(void);
void _fpreset(void)
{
    EMURESET();
}
```

FP_OFF 获取远地址偏移量

用 法 #include <dos.h>

unsigned FP_OFF (void far *farptr);

相关函数 unsigned FP_SEG (void far *farptr);

用 法 void far *MK_FP (unsigned seg, unsigned off);

原 型 在 dos.h

说 明 FP_OFF 定用于取得远指针 farptr 的偏移量。

FP_SEG 是一用于设置远指针 farptr 段值的宏。

MK_FP 是一通过段值(seg)和偏移量(off)两部分来建立一个 far 指针的宏。

返 回 值 FP_OFF 返回一个无符号整数代表偏移量。

FP_SEG 返回一个无符号整数代表段值

MK_FP 返回一个远指针

参 见 movedata, segread

FP_SEG 获取远地址段值

用 法 #include <dos.h>

unsigned FP_SEG (void far *farptr);

原 型 在 dos.h

说 明 见 FP_OFF

fprintf 传送格式化输出到一个流中

-- fprintf.c

用 法 #include <stdio.h>

int fprintf (FILE *stream, char *format[, argument, ...]);

原 型 在 stdio.h

说 明 见 printf

源 程 序

#include <stdio.h>

#include <stdio.h>

#include <_printf.h>

int cdecl fprintf (FILE *F, const char *fmt, ...)

{
 return __vprinter ((putnF *) __fputn, F, fmt, _va_ptr);
}

函 数 名 _fputc 向一个流送一个字符。

-- fputc.c

用 法 int _fputc(int ch, FILE *stream);

原型文件 stdio.h

说 明 该函数只由宏 putc() 调用。其唯一目的是在调用 fputc()之前减小层次指示器。

返 回 值 成功: 字符 CH;

失败: EOF。

源 程 序

#include <stdio.h>

#include <stdio.h>

#include <io.h>

int _fputc (char ch, FILE *fp)

{
 --fp->level;
 return(fputc(ch, fp));
}

fputc 送一个字符到一个流中

-- fputc.c

用 法 #include <stdio.h>

int fputc (int ch, FILE *stream);

原型在 stdio.h

说 明 见 putc

源 程 序

```
#include <stdio.h>
#include <stdio.h>
#include <io.h>
int fputc (int ch, register FILE *fp)
{extern int stdoutStarted;
static char cr = '\r';
register unsigned char c = ch;
putAgain:
if (++(fp->level) < 0)
{ ++fp->curp [-1] = c;
if ((fp->flags & F_LBUF) && ((c == '\n') || (c == '\r')) )
if (fflush (fp))
return EOF;
return (c);
}
fp->level --;
if (fp->flags & (_F_IN | _F_ERR) || ! (fp->flags & _F_WRIT))
{ fp->flags |= _F_ERR;
return EOF;
}
restartStdout:
fp->flags |= F_OUT;
if (fp->bsize != 0) /* is the stream buffered ? */
{ /* The level is zero only at initialization or after a rewind
or seek, when _F_OUT is not yet decided and the buffer empty. */
if (fp->level)
{ if (fflush (fp))
return EOF;
}
else
fp->level = -1 - fp->bsize;
goto putAgain;
}
else /* the stream is not buffered. */
{ if (! stdoutStarted && ((short) fp == (short) stdout))
{ if (! isatty (fp->fd))
fp->flags &= ~ F_TERM;
setvbuf (fp, NULL, (fp->flags & F_TERM) ? _IONBF : _IOFBF, BUFSIZ);
goto restartStdout;
}
if ((('\n' == c) && ! (fp->flags & F_BIN) &&
(1 != write (fp->fd, & cr, 1))) ||
(1 != write (fp->fd, & ch, 1)))
{ if ((fp->flags & F_TERM) == 0)
{ fp->flags |= _F_ERR;
return EOF;
}
}
return ((unsigned char) c);
}
}
```

fputc 送一个字符到标准输出流(stdout)中 — putc.c

用 法 int fputc (char ch);

原型在 stdio.h

说 明 见 putc

源 程 序

```
#include <stdio.h>
#include <_stdio.h>
#include <_io.h>
int fputc (register int c)
{ return putc (c, stdout);
}
```

函数名 `__fputn` - 向一个流写一些字节。 -- `putc.c`
 用法 `size_t pascal __fputn (void *ptr, register size_t n, FILE *fp)`
 原型文件 `stdio.h`
 说明 `__fputn` 将 `ptr` 所指地址的 `n` 个字节写到一个打开的流 `fp` 中。
 返回值 成功: 所写入的字节数;
 失败: 0。

源程序

```
size_t pascal __fputn (void *ptr, register size_t n, register FILE *fp)
{ int ret;
  n++;
  if (fp->flags & F LBUF)
  { while (--n && (EOF != fputc(*((char *)ptr)++, fp))) continue;
  }
  else
  { if ( (fp->flags & F BIN) && fp->bsize && (n > fp->bsize))
    { if (fp->level)
      if (flush(fp))
        return(0);
      n--;
      ret = write(fp->fd, ptr, n);
      n -= ret;
    }
    else
      while (--n && (EOF != putc(*((char *)ptr)++, fp))) continue;
  }
  return n;
}
```

`fputs` 送一个字符串到流中 -- `fputs.c`

用法 `#include <stdio.h>`
 `int fputs (char *string, FILE *stream);`

原型在 `stdio.h`

说明 见 `puts`

源程序

```
#include <stdio.h>
#include <_stdio.h>
#include <string.h>
int fputs (const register char *s, FILE *fp)
{ register int len;
  len = strlen(s);
  return __fputn (s, len, fp) ? EOF : *((unsigned char *)s+len-1);
}
```

`fread` 从一个流中读数据 -- `fread.c`

用法 `#include <stdio.h>`
 `int fread(void *ptr, int size, int nitems, FILE *stream);`

相关函数

用法 `int fwrite(void *ptr, int size, int nitems, FILE *stream);`

原型在 `stdio.h`

说明 `fread` 从指定输入流 `stream` 中读取 `nitems` 项数据, 每一项数据长度为 `size` 字节, 到由 `ptr` 所指的块中。

`fwrite` 附加 `nitems` 个数据项, 每个数据项长度为 `size` 字节到指定输出流 `stream` 中。数据从 `ptr` 处开始添加。

这两个函数, 读(写)的字节总数是 `nitems*size`。

参数说明中的 `ptr` 是指向任意对象的指针。 `size` 是 `ptr` 所指对象的大小。表达式 `sizeof*ptr` 将产生正确的值。

返回值 调用成功时, 两函数各自返回实际读或写的数据项数(非字节数)。

在遇到文件结束或出错时, `fread` 返回一短(short)计数值(可能为 0)。

`fwrite` 在出错时返回一短计数值。

可移植性 适用于所有 UNIX 系统

参见 `fopen`, `getc`, `gets`, `printf`, `putc`, `puts`, `read`, `scanf`, `write`

源程序

```
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
size_t fread (void *ptr, register size_t psize, size_t nitems, FILE *fp)
{
    register size_t n;
    unsigned long temp;
    if (! psize) return 0;
    if ( (temp = (unsigned long)psize * (unsigned long)nitems) < 0x10000L)
        return ( (unsigned)temp - _fgetn (ptr, (unsigned)temp, fp) ) / psize;
    else
    {
        n = nitems+1;
        while (--n && 0 == _fgetn (ptr, psize, fp))
            ptr = ((char huge *) ptr) + psize;
    }
    #if defined(__HUGE__)
    #pragma warn -sus
    ptr = ((char huge *) ptr) + psize;
    #pragma warn .sus
    #else
    (char *)ptr += psize;
    #endif
    return (nitems - n);
}
```

`free` 释放已分配的块

-- `free.c`

用法 `void free (void *ptr);`

原型在 `stdio.h`, `alloc.h`

说明 见 `malloc`

源程序

```
#if defined(__TINY__) || defined(__SMALL__) || defined(__MEDIUM__)
#include <stdio.h>
#include "heap.h"
#include <alloc.h>
/* adds a block to the free-block queue */
static void cdecl insert_free_block( struct header *q )
{
    struct header *n;
    if( __rover )
    {
        n = __rover->next_free;
        __rover->next_free = q;
        n->prev_free = q;
        q->next_free = n;
        q->prev_free = __rover;
    }
    else

```

```

    {
        __rover = q;
        q->prev_free = q;
        q->next_free = q;
    }
}

/* joins two adjacent free blocks together */
static void cdecl join_free_blocks( struct header *a, struct header *b )
{
    struct header *n;
    a->size += b->size;
    if( __last == b )
    {
        __last = a;
    }
    else
    {
        n = (struct header *)((char *)b + b->size);
        n->prev_real = a;
    }
    pull_free_block( b );
}

/* frees the last block in the heap and adjusts the break level */
static void cdecl free_last_block( void )
{
    struct header *p;
    if( __first == __last ) /* freeing the ONLY block? */
    {
        brk( __first );
        __first = __last = NULL;
    }
    else
    {
        p = __last->prev_real;
        if( (p->size & 0x0001) == 0 ) /* is previous block free? */
        {
            pull_free_block( p );
            if( p == __first ) __first = __last = NULL;
            else __last = p->prev_real;
            brk( p );
        }
        else
        {
            brk( __last );
            __last = p;
        }
    }
}

/* frees an interior block within the heap */
static void cdecl free_inner_block( struct header *q )
{
    struct header *n, *p;
    --q->size;
    n = (struct header *)((char *)q + q->size); /* mark it as free */
    p = q->prev_real; /* set up pointers to the */
    if( (p->size & 0x0001) == 0 && q != __first ) /* previous and next blocks */
    { /* join to the previous block? */
        p->size += q->size;
        n->prev_real = p;
        q = p;
    }
    else /* add it to the free queue */
    {
        insert_free_block( q );
    }
    if( (n->size & 0x0001) == 0 ) /* join with the next block? */
    {
        join_free_blocks( q, n );
    }
}

/* deallocates a block and returns it to the free-block queue */
void cdecl free( void *q )
{
    if ( q == NULL ) /* ANSI requires this */
        return;
    q = (char *)q - USED_HEADER_SIZE; /* calc block's start address */
    if( q == __last )
    {
        free_last_block();
    }
    else
    {
        free_inner_block( q );
    }
}

```

#endif

freemem 释放先前分配的 DOS 内存块

-- freemem.cas

用 法 int freemem(unsigned seg);

原型在 dos.h

说 明 见 allocmem

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
#include <io.h>
int freemem(unsigned segx)
{asm mov ah, 49h
asm mov cx, segx
asm int 21h
asm jc freememFailed
return(0);
freememFailed:
return KError(AX);
}
```

freopen 替换一个流

-- fopen.c

用 法 #include <stdio.h>

FILE *freopen(char *filename, char *type, FILE *stream);

原型在 stdio.h

说 明 见 fopen

源 程 序

```
FILE *freopen(const char *filename, const char *type, FILE *fp)
{
    if (fp->token != (short) fp)
        return NULL; /* validate pointer */
    fclose (fp);
    return _openfp (fp, filename, type);
}
```

frexp 把一个双精度数分解为尾数和指数

-- frexp.cas

用 法 double frexp(double value, int *epr);

原型在 math.h

说 明 见 exp

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
static int minus1 = -1;
#pragma warn -rvl
double frexp (double value, int *expP)
{
    unsigned statword;
asm FILD W0 (minus1)
asm FLD DOUBLE (value)
asm FXTRACT /* ST(1) = exponent, (pushed) ST = fraction */
asm FXCH
/* The FXTRACT instruction normalizes the fraction 1 bit higher than
   wanted for the definition of frexp() so we need to tweak the result
   by scaling the fraction down and incrementing the exponent. */
asm LES bx, expP asm FISTP W0 (ES_ [bx])
}
```

```

asm    FSCALE                /* fraction scaled as C expects */
/* if number was 0, don't touch exponent */
asm    FXAM
asm    FSTSW    statword
asm    FWAIT
asm    mov     ah, BY1(statword)
asm    sahf
asm    jz      done
asm    inc     W0 (ES_ [bx])  /* exponent biased to match */
done:
asm    FSTP    ST(1)         /* discard minus1, leave fraction as TOS */
frx_end:
    return;
}
#pragma warn .rvl

```

fscanf 从一个流中执行格式化输入 -- fscanf.c

用 法 `#include <stdio.h>`
`int fscanf(FILE *stream, char *format[,argument,...]);`

原 型 在 `stdio.h`

说 明 见 `scanf`

源 程 序

```

#include <stdarg.h>
#include <stdio.h>
#include <scanf.h>
#undef  ungetc /* remove the macro version */
int cdecl fscanf (FILE *fp, const char *fmt, ...)
{
#pragma warn -sus
    return scanner (
        (int (*)(void *)) fgetc,
        (void (*)(int ch, void *)) ungetc,
        fp,
        fmt,
        va_ptr
    );
#pragma warn .sus
}

```

fseek 重定位流上的文件指针 -- fseek.c

用 法 `#include <stdio.h>`
`int fseek(FILE *stream, long offset, int fromwhere);`

相关函数 `long ftell(FILE *stream);`

用 法 `int rewind(FILE *stream);`

原 型 在 `stdio.h`

说 明 `fseek` 设置与流 `stream` 相联的文件指针到新的位置, 该位置与 `fromwhere` 给定的文件位置的距离为 `offset` 字节。
`fromwhere` 必须是 0, 1, 2 中的一个, 分别代表以下三个符号常量(在 `stdio.h` 中定义):

<code>fromwhere</code>	文件位置
<code>SEEK_SET(0)</code>	文件开始
<code>SEEK_CUR(1)</code>	当前文件指针位置
<code>SEEK_END(2)</code>	文件末尾

`fseek` 忽略由 `ungetc` 退回的任何字符。

ftell 返回流 **stream** 中的当前文件指针位置。偏移量是从文件头开始算起的字节数。

rewind(stream)与 **fseek(stream,0,SEEK_SET)**除了以下一点外,其余是等价的:

rewind 清除文件结束标志和出错标志,而 **fseek** 只清除文件结束标志。

在调用了 **fseek** 和 **rewind** 后,在更新文件上的下一个操作可以是输入也可以是输出。

返回值 如果指针成功地移动了, **fseek** 和 **rewind** 返回零;否则返回一非零值表示失败。

ftell 在成功调用后,返回当前文件的指针位置;出错时返回-1L。

可移植性 都适用于所有的 unix 系统。

参 见 **fopen**, **ftell**, **getc**, **lseek**, **setbuf**, **ungetc**

源 程 序

```
#include <stdio.h>
#include <io.h>
int fseek(register FILE *fp, long offset, int whence)
{
    if (!flush (fp)) return EOF;
    if (SEEK_CUR == whence && fp->level > 0) offset -= Displacement (fp);
    fp->flags &= ~(_F_OUT | _F_IN | _F_EOF);
    fp->level = 0;
    fp->curp = fp->buffer;
    return (lseek (fp->fd, offset, whence) == -1L) ? EOF : 0;
}
```

fsetpos 定位流上的文件指针

-- **fsetpos.c**

用 法 **#include <stdio.h>**

int fsetpos(FILE *stream, const fpos_t *pos);

原 型 在 **stdio.h**

说 明 见 **fgetpos**

源 程 序

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos)
{
    return fseek(stream, *pos, SEEK_SET);
}
```

fstat 获取打开文件信息

-- **fstat.cas**

用 法 **#include <sys\stat.h>**

int fstat(char *handle, struct stat *buff)

原 型 在 **sys\stat.h**

说 明 见 **stat**

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <sys\stat.h>
#include <io.h> #include <_io.h>
int fstat (int fildes, struct stat *buff)
{
    asm mov     bx, fildes
    asm mov     ax, 4400h          /* IOCTL, get device information */
    asm int     21h
    asm jc      fstatFailed
    asm sub     si, si             /* SI = file mode */
    asm or      dl, dl             /* is it a character device? */
    asm js      fstatIsDevice
    /* Arrive here if the fildes is for a regular file. */
}
```

```

        SI |= S_IFREG + S_IREAD;
asm      and     dx, 3Fh          /* isolate the drive number */
asm      mov     di, dx          /* and keep it safe in DI */
fst_findSize:
        filelength (_BX);
asm      or      dx, dx
asm      jl      fstatFailed
asm      push    dx
asm      push    ax              /* save the file size */
/* File time-stamps can only be retrieved for MSDOS versions 2.0 or later. */
asm      mov     ax, 5700h       /* get date and time */
asm      int     21h
asm      jc      fst_zeroTime
/* MSDOS time is a 32-bit record, which must be converted into the Unix
   style of seconds since 1970. */
asm      push    dx              /* date */
asm      push    cx              /* time */
asm      call    EXTPROC (__DOStimeToU)
asm      xchg    cx, ax          /* result in DX:CX */
asm      jmp     short fst_construct
/* Arrive here if IOCTL reported the handle to be for a character device. */
fst_isDevice:
        SI |= S_IFCHR + S_IREAD + S_IWRITE;
asm      mov     di, bx          /* use the files as device number */
asm      sub     ax, ax
asm      push    ax
asm      push    ax              /* zero length, on stack */
fst_zeroTime:
asm      sub     cx, cx
asm      mov     dx, cx
/* Arrive here with SI = mode, DI = device, DX:CX = time, and
   the file length on stack. */
fst_construct:
asm      xchg    ax, di
asm      LES     di, bufP
#ifdef LDATA
asm      push    DS
asm      pop     ES
#endif
asm      cld
asm      stosw   bx, ax          /* device */
asm      xchg    bx, ax          /* keep a copy */
asm      sub     ax, ax
asm      stosw   /* inode */
asm      xchg    ax, si
asm      stosw   /* mode */
asm      mov     ax, 1
asm      stosw   /* number of links */
asm      xchg    ax, si          /* bring back the zero */
asm      stosw   /* user (owner) id */
asm      stosw   /* group id */
asm      xchg    ax, bx
asm      stosw   /* real device */
asm      pop     ax
asm      stosw   /* file.. */
asm      pop     ax
asm      stosw   /* ..size */
asm      xchg    ax, cx
asm      stosw
asm      xchg    ax, dx
asm      stosw   /* access time */
asm      xchg    ax, dx
asm      stosw   /* modification time */
asm      xchg    ax, dx
asm      stosw
asm      xchg    ax, dx
asm      stosw   /* status change time */

```

```

fst_end:
    return 0;
fstatFailed:
    return __IOerror (_AX);
}

```

ftell 返回当前文件指针

-- fseek.c

用 法 #include <stdio.h>

```
long ftell(FILE *stream);
```

原 型 在 stdio.h

说 明 见 fseek

源 程 序

```

#include <stdio.h>
#include <io.h>
long ftell(register FILE *fp)
{
    long a;
    if (!flush (fp)) return -1L;
    a = lseek(fp->fd, 0L, SEEK_CUR);
    return (fp->level > 0) ? a - Displacement (fp) : a;
}

```

ftime 把当前时间存到 timeb 结构中

-- ftime.c

用 法 #include <sys\timeb.h>

```
void ftime(struct timeb *buf);
```

原 型 在 sys\timeb.h

说 明 ftime 函数把当前时间存到 buf 所指的 timeb 结构中。该结构内含从 1970 年 1 月 1 日以来的时间秒值和独立的毫秒值。同时也含有地方时区和夏令时标志。结构 timeb 在 sys\timeb.h 中定义如下:

```

struct timeb {
    long time; /* second since 1/1/1970 */
                /* Greenwich Mean Time */
    short millitm; /* fraction of second */
                /* ( in milliseconds) */
    short timezone; /* difference between local time and GMT */
    short dstflag; /* if daylight savings time is not in effect */
};

```

源 程 序

```

#include <sys\timeb.h>
#include <time.h>
#include <dos.h>
#include <mem.h>
#include <io.h>
void ftime(struct timeb *TimeStructPtr)
{
    struct date DosDate, Vdate;
    struct time DosTime;
    tzset(); /* Get timezone info. */
    /* Because there is a window of vulnerability at exactly midnight when
       calling getdate(INT 21 fn 2A) and gettime(INT 21 fn 2C) in succession.
       we do 2 calls to getdate() to ensure we aren't in this window. */
    do
    {
        getdate(&DosDate);
        gettime(&DosTime);
        getdate(&Vdate);
    }
    while ( (DosDate.da_year != Vdate.da_year) ||
            (DosDate.da_day != Vdate.da_day) ||
            (DosDate.da_mon != Vdate.da_mon) );
    /* Convert external timezone's seconds to structure timezone's minutes.

```

```

        Set daylight savings indicator.
        Convert DOS date and time to UNIX style time and store in structure.
        Set milliseconds structure field. DOS is only accurate to 100ths of a
        second so (100ths * 10) makes 1000ths. */
TimeStructPtr->timezone = timezone / 60;
if (daylight && __isDST( DosTime.ti_hour, DosDate.da_day,
                        DosDate.da_mon, DosDate.da_year-1970) )
    TimeStructPtr->dstflag = 1;
else
    TimeStructPtr->dstflag = 0;
TimeStructPtr->time = dostounix(&DosDate, &DosTime);
TimeStructPtr->millitm = DosTime.ti_hund * 10;
}

```

fwrite 写内容到流中 -- fwrite.c

用 法 #include <stdio.h>

```
int fwritew(void *ptr, int size, int nitems, FILE *stream);
```

原 型 在 stdio.h

说 明 见 fread

源 程 序

```

#include <stdio.h>
#include <_stdio.h>
size_t fwrite(const void *ptr, register size_t psize, size_t nitems,
              FILE *fp)
{
    register size_t n;
    unsigned long temp;
    if (! psize) return nitems;
    if ( (temp = (unsigned long)psize * (unsigned long)nitems) < 0x10000L)
        return ( (unsigned)temp - __fputn (ptr, (unsigned)temp, fp) ) / psize;
    else
    {
        n = nitems+1;
        while (--n && 0 == __fputn (ptr, psize, fp))
            ptr = ((char huge *) ptr) + psize;
        #if defined(_HUGE_)
        #pragma warn -sus
        ptr = ((char huge *) ptr) + psize;
        #pragma warn .sus
        #else
        (char *) ptr += psize;
        #endif
        return (nitems - n);
    }
}

```

gcvt 把浮点数转换成字符串 -- gcvt.c

用 法 #include <dos.h>

```
chsr *gcvt(double value, int ndigit, char *buf);
```

原 型 在 stdlib.h

说 明 见 ecvt

源 程 序

```

#include <stdlib.h>
#include <printf.h>
char *gcvt (double value, int ndec, char *bufP )
{
    __realtvt (&value, ndec, bufP, 'g', 0, F_8byteFloat);
    return bufP;
}

```

geninterrupt 产生软中断

用 法 #include <dos.h>

`void geninterrupt(int intr_num);`

原型在 `dos.h`

说明 见 `disable`

函数名 `Get` - 获取字符串中下一字符。 -- `strtoi.c`

用法 `static int Get(char **strPP);`

返回值 字符串中的下一字符。如果下一字符为空，则返回-1。

源程序

```
#include <stdlib.h>
#include <limits.h>
#include <ctype.h>
#include <errno.h>
#include <scanf.h>
#include <stddef.h>
static int Get(char **strPP)
{
    register unsigned c;
    return ((c = *((*strPP) ++)) == 0) ? -1 : c;
}
```

`getarccoords` 取得最后一次调用 `arc` 的坐标

用法 `#include <graphics.h>`

`void far getarccoords(struct arccoordstype far *arccoords);`

原型在 `graphics.h`

说明 见 `arc`

`getaspectratio` 返回当前图形模式的纵横比

用法 `#include <graphics.h>`

`void far getaspectratio(int far *xasp, int far *yasp);`

原型在 `graphics.h`

说明 见 `arc`

`getbkcolor` 返回当前背景颜色

用法 `#include <graphics.h>`

`int far getbkcolor(void);`

相关函数

用法 `void far setbkcolor(int color);`

原型在 `graphics.h`

说明 `getbkcolor` 返回当前背景颜色(详见下表说明)。

`setbkcolor` 将背景设计成参数 `color` 所指定的颜色，参数 `color` 可以是名字或是数字，如下表所列（符号在 `GRAPHICS.H` 中定义）：

数值	名字	数值	名字
0	BLACK	1	BLUE
2	GREEN	3	CYAN
4	RED	5	MAGENTA
6	BROWN	7	LIGHTGRAY

8	DARKGRAY	9	LIGHTBLUE
10	LIGHTGREEN	11	LIGHTCYAN
12	LIGHTRED	13	LIGHTMAGENTA
14	YELLOW	15	WHITE

例如：如果想把背景颜色设置为蓝色，可以调用：

```
setbkcolor(BLUE)
```

或者

```
setbkcolor(1)
```

在 CGA 和 EGA 系统中，setbkcolor 的参数指出了当前调色板的入口项。可通过改变调色板的第一个入口点来改变背景颜色。

返回值 getbkcolor 返回当前背景颜色

setbkcolor 返回空

可移植性 在 Turbo Pascal 5.0 中有相应的子程序

参 见 getpalette, initgraph

getc 从流中取字符

-- getch.cas

用 法 #include <stdio.h>

```
int getc(FILE *stream);
```

相关函数 int fgetc(FILE *stream);

用 法 int fgetchar(void);

```
int getch(void);
```

```
int getchar(void);
```

```
int getche(void);
```

```
int getw(FILE *stream);
```

```
int ungetc(char c, FILE *stream);
```

```
int ungetch(int c);
```

原 型 在 stdio.h

```
conio.h(getch, getche, ungetch)
```

说 明 getc 是一个返回指定输入流 stream 中下一个字符的宏。

getchar 是一个定义为 getc(stdin) 的宏

ungetc 把字符 c 退回到指定名的输入流 stream 中。该字符在下次对流 stream 调用 getc 或 fread 时被返回。在所有情况下，都只能退回一个字符。在未调用 getc 的情况下再次调用 ungetc 将使前一个字符失去。fseek 清除所有被退回的字符。

fgetc 与 getc 功能相同，只是它是一函数而 getc 是一宏。

fgetchar 是一个相当于 fgetc(stdin) 的宏

getch 是一个从控制台直接读一字符，而无回显的函数。

getche 是一个从控制台读并回显一个字符的函数。

ungetch 把字符 c 退回到控制台，使其成为下一个所读字符。在调用下次 read 以前，如果多次调用 ungetch，它将失败。

getw 返回指定名输入流 stream 中的下一个整数，它假定在文件中无特殊的对齐字符。

返回值 在成功调用时，getc, getchar, fgetc 和 fgetchar 返回所读的字符，它已被转换为无符号扩展的整形值。在遇到文件结束或出错时，它们都返回 EOF。

getch 和 getche 返回所读的字符，这两个函数没有出错返回。

getw 返回输入流 stream 中的下一个整数。在遇到文件结束或出错标志时，它返回 EOF。因为 EOF 是 getw 返回的合法值，因此应用 feof 或 ferror 来检测是文件结束还是出错。

ungetc 总是返回被退回的字符。

ungetch 在调用成功时返回字符 C；如果返回为 EOF，表示出错。

可移植性 getch, getche 和 ungetche 只适用于 MS-DOS。

fgetc, fgetchar, getc, getchar, getw 和 ungetc 适用于 UNIX 系统。

getc 和 getchar 在 kernighan 和 Ritchie 所著书中定义。

参 见 ferror, fopen, fread, fseek, gets, putc, read, scanf

源 程 序

```
#pragma inline
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#ifdef TC_OS2
#include <os2tc.h>
#endif
static unsigned char _cFlag = 0; /* Flag presence of un-gotten char */
static unsigned char _cChar = 0; /* The ungotten char */
/* if these routines are split into separate files, each
   must extern char _unflag & _unchar, _unflag & _unchar must not be static. */
int getch(void)
{
    if (_cFlag)
    {
        _cFlag = 0; /* Reset the flag */
        _AL = _cChar; /* Get the un-got char */
    }
#ifdef TC_OS2
    else
    {
        struct KeyData sKD;
        /*In protected mode it's possible for getch() to fail. This
           happens if the app is detached. We'll return -1 if it fails. */
        return (KBDCHARIN(
            (struct KeyData far *) &sKD, /* Return struc */
            (WORD) 0, /* WAIT for key */
            (WORD) 0 /* Kbd handle */
        ))
        ? -1 : (int)(unsigned)sKD.char_code;
    }
#else
    else
    {
        _AX = 0x0700; /* Console input (no echo) */
        _geninterrupt(0x21);
    }
#endif
    _AH = 0; /* Force it to be unsigned */
    return _AX;
}
#ifdef __OLDCONIO__
int getche(void)
{
    if (_cFlag)
    {
        _cFlag = 0; /* Reset the flag */
        _AL = _cChar; /* Get the un-got char */
    }
#ifdef TC_OS2
    else
    {
        struct KeyData sKD;
        unsigned char cChar;
        /*In protected mode it's possible for getche() to fail. This
           happens if the app is detached. We'll return -1 and echo
           nothing if it fails.
           There are some undesirable aspects of using KbdStringIn which
           would do the echo for us, it doesn't allow BINARY reads if
           using echo mode. So... we'll do a KbdCharIn and VioWrTtTy. */
        if (KBDCHARIN(
```

```

        (struct KeyData far *) &sKD,      /* Return struc */
        (WORD) 0,                        /* WAIT for key */
        (WORD) 0                          /* Kbd handle */
    ))
    return -1;
else
    VIOWRTITY(
        (CFAR) &sKD.char_code, /* The char */
        (WORD) 1,              /* Length=1 */
        (WORD) 0                /* VIO handle */
    );
    _AL = sKD.char_code;
;
#else
    else
    {
        _AX = 0x0100;          /* Console input (echo) */
        geninterrupt(0x21);
    }
#endif
    _AH = 0;                    /* make it unsigned */
    return _AX;
}
#else
int getche(void)
{
    int ch;
    if (_cFlag)                /* Prevent possible double echoing */
        ch = getch();
    else
        putchar(ch = getch());
    return ch;
}
#endif
int ungetch(int c)
{
    if (_cFlag)
        return(EOF);
    _cFlag = 1;
    _AX = c;
    _cChar = _AL;
    return _AX;
}

```

getcbkr 获取 Control_break 设置

-- getcbkr.c

用 法 int getcbkr(void);

相关函数

用 法 int setcbkr(int value);

原 型 在 dos.h

说 明 getcbkr 使用 MS-DOS 系统调用 0x33 返回 CONTRAL_BREAK 检测的当前设置。

setcbkr 使用 MS-DOS 系统调用 0x33 设置 CONTRAL_BREAK 的检测状态为 ON 或 OFF。

value=0 置检测状态为 OFF(只在控制台、打印机或通信设备进行 T/O 时检测)

value=1 置检测状态为 ON(每次系统调用时都检测)

返 回 值 如果 CONTRAL_BREAK 检测状态为 off, getcbkr 返回 0; 如果检测状态为 ON, 返回 1。

setcbkr 返回 value

可移植性 只适用于 MS-DOS

源 程 序

```

#include <dos.h>
int getcbkr(void)
{
    _AX = 0x3300;
    geninterrupt(0x21);
    return(_DL);
}

```


getch 从控制台无回显地取一字符

-- getch.cas

用 法 int getch(void);

原 型 在 conio.h

说 明 见 getc

源 程 序

```
#pragma inline
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#ifdef TC_OS2
#include <os2tc.h>
#endif
static unsigned char _cFlag = 0; /* Flag presence of un-gotten char */
static unsigned char _cChar = 0; /* The ungotten char */
/* if these routines are split into separate files, each
   must extern char _unflag & _unchar, _unflag & _unchar must not be static. */
int getch(void)
{
    if (_cFlag)
    {
        _cFlag = 0; /* Reset the flag */
        _AL = _cChar; /* Get the un-got char */
    }
#ifdef TC_OS2
    else
    {
        struct KeyData sKD;
        /* In protected mode it's possible for getch() to fail. This
           happens if the app is detached. We'll return -1 if it fails. */
        return (KBDCHARIN(
            (struct KeyData far *) &sKD, /* Return struc */
            (WORD) 0, /* WAIT for key */
            (WORD) 0 /* Kbd handle */
        ))
        ? -1 : (int)(unsigned)sKD.char_code;
    }
#else
    else
    {
        _AX = 0x0700; /* Console input (no echo) */
        _geninterrupt(0x21);
    }
#endif
    _AH = 0; /* Force it to be unsigned */
    return _AX;
}
```

getchar 从流中取字符

用 法 #include <stdio.h>

int getchar(void);

原 型 在 stdio.h

说 明 见 getc

getche 从控制台中取一字符，并回显

用 法 int getche(void);

原 型 在 conio.h

说 明 见 getc

getcolor 返回当前的画线颜色

用 法 `#include <graphics.h>`
`int far getcolor(void);`

相关函数

用 法 `void far setcolor(int color);`

原 型 在 `graphics.h`

说 明 `getcolor` 返回当前画线颜色。

`setcolor` 将当前画线颜色置为 `color`，参数 `color` 的取值范围为零到 `getmaxcolor()`。

画线颜色是指在画线或其它图形时像素被设置的值。例如，在 CGA 模式中，调色板包含四种颜色：背景颜色、淡绿色、淡红色和黄色。在此模式下，如果 `getcolor` 果 `getcolor()` 返回 1，则当前画线颜色为淡绿色；类似地，`setcolor(3)` 选择黄色作为画线颜色。

返 回 值 `getcolor` 返回当前画线颜色，`setcolor` 无返回值。

可移植性 Turbo Pascal 5.0 中有相似的子程序

参 见 `getpalette`, `getmaxcolor`

getcurdir 取指定驱动器的当前目录 -- `getcurdi.cas`

用 法 `int getcurdir(int drive, char *direc);`

原 型 在 `dir.h`

说 明 `getcurdir` 取指定驱动器 `drive` 的当前工作目录。

`drive` 包含一驱动器号(0=缺省值, 1=A 等)

`direc` 指向一个长度为 `MAXDIR` 的存储区，用于存放目录名。空字符结束的名不包含驱动器号，也不以反斜杠开始。

返 回 值 在调用成功时，`getcurdir` 返回 0；在出错时返回-1。

可移植性 只适用于 MS-DOS 系统

参 见 `freegetcwd`

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <i.h>
int
getcurdir(int drive, char *direc)
{
    pushDS
    asm mov     ah, 047h
    asm mov     di, drive
    asm LDS     si, direc
    asm int     021h
    popds
    if getcurdirFailed
        return 0;
getcurdirFailed:
    return _IOerror(_AX);
}
```

getcwd 取当前工作目录 -- `getcwd.c`

用 法 `char *getcwd(char *buf, int n);`

原 型 在 `dir.h`

说 明 `getcwd` 取 `cwd`(当前工作目录，包括驱动器号)的完整路径名，最长为 `n` 个字节，并把它存在缓冲区 `buf` 中。如果该完整路径名长度(包括空字符终结符)长度超过 `n`，则出错。

如果 `buf` 为 `NULL`，则可用 `malloc` 分配一个 `n` 字节长的缓冲区，以后可以通过传送 `getcwd` 的返回值到 `free` 函数的方法释放该缓冲区。

返回值 `getcwd` 返回 `buf`。在发生错误时, 返回 `NULL`, 此时, 全局变量 `errno` 被置为下列值之一:

<code>ENOENT</code>	无此设备
<code>ENOMEM</code>	无足够存储区
<code>ERANGE</code>	结果超出范围

可移植性 只适用于 MS-DOS

参 见 `free`, `getcurdir`, `malloc`

源 程 序

```
#include <dir.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <alloc.h>
char *getcwd(char *bufp, int bufL)
{
    char    buf1[MAXDIR + 2];
    /* Get current disk */
    buf1[0] = getdisk() + 'A';
    buf1[1] = ':';
    buf1[2] = '\\';
    /* Get current directory in a work buffer */
    if (getcurdir(0, &buf1[3]) == -1)
        return NULL;
    if (strlen(buf1) >= bufL)
    {
        errno = ERANGE;
        return NULL;
    }
    /* Allocate a buffer if bufp is NULL */
    if (bufp == NULL)
        if ((bufp = malloc(bufL)) == NULL)
        {
            errno = ENOMEM;
            return NULL;
        }
    strcpy(bufp, buf1);
    return bufp;
}
```

`getdate` 取 MS-DOS 日期

-- `getdate.c`

用 法 `#include <dos.h>`

`void getdate(struct *dateblk);`

相关函数 `void gettime(struct time *timep);`

用 法 `void setdate(struct date *dateblk);`

`void settime(struct time *timep);`

原 型 在 `dos.h`

说 明 `getdate` 把系统当前日期填入 `date` 结构中(由 `dateblk` 所指)。

`gettime` 把系统当前时间填入由 `timep` 所指 `time` 结构中。

`setdate` 设置系统日期(月, 日, 年)为 `dateblk` 所指的 `date` 结构的值。

`settime` 设置系统时间为 `timep` 所指 `time` 结构中的值。

`date` 结构如下:

```
struct date{
    int da_year; /*current year*/
    char da_day; /*Day of the month */
    char da_mon; /*Month (1 =jan) */
};
```

time 结构定义如下:

```
struct time{
    unsigned char ti_min; /*minutes*/
    unsigned char ti_hour; /*hour */
    unsigned char ti_hund; /*hundredth of seconds*/
    unsigned char ti_sec; /*seconds*/
}
```

返回值 这些函数不返回值

可移植性 只适用于 MS-DOS

参 见 ctime

源 程 序

```
#include <dos.h>
void getdate(struct date *datep)
{
    _AH = 0x2a;
    _geninterrupt(0x21);
    ((int *)datep)[0] = _CX;
    ((int *)datep)[1] = _DX;
}
```

getdefaultpalette 返回指向调色板定义结构的指针

用 法 #include <graphics.h>

struct palettetype *far getdefaultpalette(void);

说 明 本函数返回一指针, 该指针指向调用 initgraph 初始化程序时当前驱动程序的调色板结构。

结构 palettetype 在 graphics.h 中定义如下:

```
#define MAXCOLRS 15
struct palettetype {
    unsigned char size;
    signed char color[MAXCOLORS+1]
}
```

原 型 在 graphics.h

参 见 getpalette, initgraph

getdfree 取磁盘自由空间 -- getfat.cas

用 法 #include <dos.h>

void getdfree(int drive, struct dfree *dfreep);

原 型 在 dos.h

说 明 getdfree 接受磁盘驱动器号 drive(0=缺省, 1=A 等), 并把磁盘特性填入由 dfreep 所指的 dfree 结构中。

dfree 结构定义如下:

```
struct dfree{
    unsigned df_avail; /*available clusters*/
    unsigned df_total; /*TOTAL CLUSTERS*/
    unsigned df_bsec; /*Bytes per sector */
    unsigned df_sclus; /*sectors per cluster*/
}
```

返回值 getdfree 不返回值, 出错时, dfree 结构中的 df_sclus 域被置为-1。

可移植性 只适用于 MS-DOS

参 见 getfat

源 程 序

```
void getdfree(unsigned char drive, struct dfree *dtable)
{
    int i;
    asm mov ah, 036h
    asm mov dl, drive
    asm int 021h
    i = _BX;
    ((int *)dtable)[3] = _AX;
    ((int *)dtable)[0] = i;
    ((int *)dtable)[1] = _DX;
    ((int *)dtable)[2] = _CX;
}
```

getdisk 取当前磁盘驱动器号

-- chdir.cas

用 法 int getdisk(void);

相关函数

用 法 int setdisk(int drive);

原 型 在 dir.h

说 明 getdisk 取当前磁盘驱动器号，并返回一整数：0=A:，1=B:，2=C:等。（等价于 DOS 功能 0X19）关于如何使用 getdisk 的例子，请参阅 getcurdir。

setdisk 设置当前磁盘驱动器号，0=A:，1=B:，2=C:等。（等价于 DOS 功能 0X16）

返 回 值 getdisk 返回当前驱动器号。

setdisk 返回可用驱动器数。

可移植性 只适用于 MS-DOS

参 见 getcurdir

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <io.h>
int getdisk(void)
{
    asm mov ah, 019h
    asm int 021h
    asm xor ah, ah
    return _AX;
}
```

getdrivename 返回指向当前驱动程序名的指针

用 法 #include <graphics.h>

char *getdrivename(void);

原 型 在 graphics.h

说 明 本函数调用后返回一指针，该指针指向含有当前图形驱动程序名的字符串，可以用它来测试硬件适配器。

参 见 initgraph

getdta 取磁盘传输地址

-- getdta.cas

用 法 char far *getdta(void);

相关函数

用 法 void setdta(char far *dta);

原型在 dos.h

说明 getdta 返回磁盘传输地址(DTA)的当前设置。

在小型的中型模式中,假定段为当前数据段,若仅仅用C语言,就是这种情况。但如果使用汇编子程序,它可以设置磁盘传输地址为任意硬件地址。

在紧缩、大型或巨型存储模式中, getdta 返回的地址是正确的硬件地址。该地址可以在程序外面。

setdta 改变当前 DTA 设置为 DTA 所指的值。

返回值 getdta 返回一指向传输地址的指针

setdta 没有返回值

可移植性 只适用于 MS-DOS

源程序

```
#pragma inline
#include <dos.h>
char far *getdta(void) {
asm    mov    ah, 02Fh
asm    int    021H
    return (char _es *) _BX;
}
```

getenv 从环境中取字符串

-- getenv.cas

用法 char *getenv(char *envvar);

相关函数

用法 int putenv(char *envvar);

原型在 stdlib.h

说明 MS-DOS 环境由下列形式的一系列项目组成:

envvar = varvalue

getenv 检索环境表中与 envvar 相对应的入口项,然后返回指向 varvalue 的指针。

putenv 接受字符串 envvar,把它加到当前环境中,它也可用于修改或删除一个已存在的环境变量 envvar,删除一个已存在项可以通过使 varvalue 为空来实现,如"MYVAR="。

返回值 在调用成功时, getenv 返回指向与 envvar 相联的指针,此指针在以后的调用中被重写。如果指定的 envvar 在环境中没有定义, getenv 返回零。

在调用成功时, putenv 返回 0; 失败时返回-1。

可移植性 适用于 unix 系统

参见 environ(变量),getdfree

源程序

```
#pragma inline
#include <asmrules.h>
#include <stdlib.h>
#include <stddef.h>
char *getenv(const char *nameP)
{ char **envP;
  /* Compute nameP length and remember first character */
asm    LES    di, nameP
#ifdef (LDATA)
asm    mov    ax, ES
asm    or     ax, di
#else
asm    push    ds
asm    pop     es
asm    or     di, di
#endif
asm    jz     VarNotFound
```

```

asm          mov     al, 0
asm          mov     ah, ES_ [di]    /* remember first char of name */
asm          mov     cx, -1
asm          cld
asm          repne   scasb
asm          not     cx
asm          dec     cx
asm          jz      VarNotFound
asm          /* Look for nameP in environment array */
#if _HUGE_
asm          mov     di, seg environ
asm          mov     DS, di
#endif
asm          LES     di, DPTR_(environ)
#if (LDATA)
asm          mov     W1 (envP), ES
asm          mov     bx, ES
asm          or      bx, di
#else
asm          or      di, di
#endif
asm          mov     W0 (envP), di
asm          juz     FirstVariable
/* If no match can be found, return a NULL pointer. */
VarNotFound:
    return NULL;
    /* Does the first character match ??? */
NextVariable:
asm          add     W0 (envP), dPtrSize
asm          LES     di, envP
FirstVariable:
asm          LES     di, ES_ [di]
#if (LDATA)
asm          mov     bx, ES
asm          or      bx, di
#else
asm          or      di, di
#endif
asm          jz      VarNotFound
asm          mov     al, ES_ [di]
asm          or      al, al          /* "" terminates environment */
asm          jz      VarNotFound
asm          cmp     ah, al          /* compare first characters */
asm          jne     NextVariable
asm          mov     bx, cx
asm          cmp     BY0 (ES_ [di+bx]), '='
asm          jne     NextVariable    /* must be followed by '=' */
asm          /* Yes, so compare the remainder of nameP */
asm          pushDS
asm          LDS     si, nameP
asm          repe    cmpsb
asm          popDS
asm          xchg     cx, bx
asm          jne     NextVariable
asm          inc     di
/* Now return the pointer to the caller. */
VarFound:
#if (LDATA)
    return (char _es *) _DI;
#else
    return (char *) _DI;
#endif
}

```

getfat 取文件分配表信息

-- getfat.cas

用 法 #include <dos.h>

```
void getfat(int drive, struct fatinfo *fatblkp);
```

相关函数

用 法 void getfatd(struct fatinfo *fatblkp);

原 型 在 dos.h

说 明 getfat 返回由 drive(0=缺省, 1=A:, 2=B:等)指定的驱动器和文件分配表信息。fatblkp 指向待填入内容的 fatinfo 结构。

getfatd 除了总是使用缺省驱动器号(0)外,其余同 getfat 一样。

由 getfat 和 getfatd 使用的 fatinfo 结构定义如下:

```
struct fatinfo{
    char fi_sclus; /*Sectors per cluster */
    char fi_fatid; /*The FAT id byte*/
    int fi_nclus; /*Number of clusters*/
    int fi_bysec; /*Bytes per sector */
}
```

返 回 值 无

可移植性 只适用于 MS-DOS

参 见 getdfree

源 程 序

```
#pragma inline
#include <dos.h>
void getfat(unsigned char drive, struct fatinfo *dtable)
{
    #if !defined(_HUGE_)
    asm    push    ds
    #endif
    asm    mov     ah, 01ch
    asm    mov     dl, drive
    asm    int     021h
    asm    mov     ah, [bx]
    #if !defined(_HUGE_)
    asm    pop     ds
    #endif
    ((int *)dtable)[0] = _AX;
    ((int *)dtable)[1] = _DX;
    ((int *)dtable)[2] = _CX;
}
```

getfatd 取文件分配表信息

-- getfat.cas

用 法 #include <dos.h>

void getfatd(struct fatinfo *fatblkp);

原 型 在 dos.h

说 明 见 getfat

源 程 序

```
void getfatd(struct fatinfo *dtable)
{
    #if !defined(_HUGE_)
    asm    push    ds
    #endif
    asm    mov     ah, 01bh
    asm    int     021h
    asm    mov     ah, [bx]
    #if !defined(_HUGE_)
    asm    pop     ds
    #endif
    ((int *)dtable)[0] = _AX;
```



```
((int *)dtable)[1] = _DX;
((int *)dtable)[2] = _CX;
```

getfillpattern 将用户定义的填充模式拷贝到存储区

用 法 `#include <graphics.h>`

`void far getfillpattern(char far *upattern)`

相关函数

用 法 `void far setfillpattern(char far *upattern,int color);`

说 明 **getfillpattern** 把用户定义的填充模式拷贝到由 **upattern** 所指的 8 字节存储区, 此用户定义模式用 **setfillpattern** 设置。**setfillpattern** 同 **setfillstyle** 除了 **setfillpattern** 设置用户定义的 8X8 模式而不是预定义模式这一点外, 其余是相同的。

upattern 是一指向连续 8 字节的指针, 每个字节与该模式下的 8 个象素相对应。只要其中某个模式的某一个位置被置为 1, 则对应象素将被画出来。如以下的用户定义填充模式代表一个检测板:

```
char checkboard[8]={
0XAA,0X55,0xAA,0X55,0XAA,0X55,0XAA,0X55
};
```

返 回 值 无

可移植性 在 Turbo Pascla 5.0 中有相似的子程序

参 见 **getfillsettings**

getfillsettings 取得有关当前填充模式和填充颜色的信息

用 法 `#include <graphics.h>`

`void far getfillsettings(struct fillsettingstype far *fillinfo);`

相关函数

用 法 `void far setfillstyle(int pattern,int color);`

说 明 函数 **bar**, **bar3d**, **fillpoly**, **floodfill** 和 **pieslice** 用当前填充模式和当前填充颜色来填充一个区域。有 11 种预定义模式的符号名由 **GRAPHICS.H** 中的枚举型 **fill_patterns** 提供(见下表)。另外, 用户还可以定义自己的填充模式。

getfillsettings 将有关当前填充模式和填充颜色的信息填进由 **fillinfo** 所指的 **fillsettingstype** 结构中, 此结构在 **GRAPHICS.H** 中定义如下:

```
struct fillsettingstype {
int pattern /*current fill pattern */
int color; /*current fill color */
};
```

如果 **pattern=12(USER_FILL)**, 则使用用户定义的填充模式; 否则由参数 **pattern** 给出预定义模式号。

setfillstyle 设置当前填充模式和填充颜色。如果想设置用户定义的填充模式, 则不是将 **setfillstyle** 的 **pattern** 值取 12, 而应该调用 **setfillpattern**。

在 **GRAPHICS.H** 中定义的枚举类型 **fill_patterns** 给出预定义填充模式常量, 并为用户自行定义模式提供一个指示标志:

名 称	值	描述
EMPTY_FILL	0	用背景颜色填充

SOLID_FILL	1	单色填充
LIN_FILL	2	用一填充
LTSLASH_FILL	3	用///填充
SLASH_FILL	4	用粗线\\\填充
BKSLASH_FILL	5	用粗线\\\填充
LTBKSLASH_FILL	6	用\\\填充
HATCH_FILL	7	用淡影线填充
XHATCH_FILL	8	用交叉线填充
INTERLEAVE_FILL	9	用间隔线填充
WINDDOT_FILL	10	用稀疏空白点填充
CLOSEDOT_FILL	11	用密集空白点填充
USER_FILL	12	用户定义的填充模式

除了 EMPTY_FILL 使用当前背景颜色外, 其余所有模式均使用当前填充颜色填充。

返回值 无。

如果把无效的输入传给了 setfillstyle, graphresult 将返回-11, 当前填充模式和填充颜色保持不变。

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 arc, bar, fillpoly, floodfill, getfillpattern

getfp 获取文件指针。 -- fopen.c

用 法 static FILE * pascal near getfp(void)

原型文件 局限于本模块

说 明 获取文件指针。

源 程 序

```
static FILE *pascal near getfp(void)
{
    register FILE *fp;
    fp = streams;
    while (fp->fd >= 0 && fp++ < _streams + OPEN_MAX)
        continue;
    if (fp->fd >= 0)
        return NULL;
    else
        return fp;
}
```

getftime 取文件日期和时间 -- getftime.cas

用 法 #include <dos.h>

int getftime (int handle, struct ttime *ftimep);

相关函数

用 法 int setftime(int handle, struct ttime *ftimep);

原 型 在 dos.h

说 明 getftime 取与打开文件句柄 handle 相关联的磁盘文件的时间和日期。文件的时间和日期存在由 ftimep 所指的 ttime 结构中。

setftime 把与打开文件句柄 handle 相联的磁盘文件的时间和日期存于由 ftimep 所指的 ttime 结构中。

ttime 的结构定义如下:

```

struct ftime {
    unsigned ft_tsec:5; /*Two second */
    unsigned ft_min:6; /*Minutes */
    unsigned ft_hour:5; /*Hours*/
    unsigned ft_day:5; /*Days*/
    unsigned ft_month:4; /*Moths*/
    unsigned ft_year:7; /*Year - 1980 */
}

```

返回值 在调用成功时, 这两个函数都返回 0。

在出错时, 返回-1, 并置全局变量。error 为下列值之一:

EINVFNC	无效功能号
EBADF	无效文件号

可移植性 只适用于 MS-DOS

参 见 fread

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
int getftime(int fd, struct ftime *ftimep)    (
asm    mov    ax, 05700h
asm    mov    bx, fd
asm    int    021h
asm    jc     getftimeFailed
asm    LES    di, ftimep
asm    mov    ES [di], cx
asm    mov    ES [di+2], dx
asm    return (0);
getftimeFailed:
    return __IOerror (_AX);
)

```

getgraphmode 返回当前图形模式

用 法 #include <graphics.h>
int far getgraphmode(void);

相关函数

用 法 void far setgraphmode(int mode);

原 型 在 graphics.h

说 明 getgraphmode 返回由 initgraph 或 setgraphmode 所设置的当前图形模式。

setgraphmode 选择一个不同于 initgraph 所设置缺省模式的图形模式。参数 mode

对于当前设备驱动器来说必须是一个合法的模式。setgraphmode 清除屏幕, 将所有图形设置为它们的缺省值(CP、调色板、颜色、视区等等)。可以使用 setgraphmode 和 restorecrtmode 来进行文本与图形模式之间的转换。

程序在使用这些函数之前必须首先成功地调用了 initgraph。

在 GRAPHICS.H 中定义的枚举类型 graphics_modes 给出了预定义图形模式的名字。可参阅有关的 initgraph 的说明, 那里有一张表格列出了这些枚举值。

返回值 无。

如果给了 setgraphmode 一个对当前驱动器来说无效的模式, grapresult 返回 -10。

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

getimage 将指定区域的一个位图象存到主存储区中

用 法 `#include <graphics.h>`

```
void far getimage(int left,int top, int right, int bottom,
                  void far *bitmap);
```

相关函数 `unsigned far imagesize(int left,int top,int right, int bottom);`

用 法 `void far putimage(int left, int top, void far *bitmap, int op);`

原 型 在 `graphics.h`

说 明 这三个函数用于把屏幕上的一个图象拷贝到主存储区，然后把它放回屏幕。

getimage 将屏幕上一个矩形区域的位象图存到主存储区中。`left,top,right` 和 `bottom` 四个参数用于定义屏幕上的矩形。`bitmap` 指向主存储区中存放位图象的区域。该区域的前两个字节用于存放矩形的宽和高，其余部分存放图象本身。

imagesize 决定 **getimage** 用于保存指定矩形所需的字节数。它返回的图象大小包括用于记录矩形宽和高的空间。

putimage 将以前用 **getimage** 保存的位图象重新送回屏幕。图象的左上角位于(`left,top`)。`bitmap` 指向保存源图象的主存储区域。

putimage 的参数 `OP` 指明了一个组合算子，它用于控制计算象素的颜色。这是基于已在屏幕上的象素和主存储区中对应的源象素来计算的。

在 `GRAPHICS.H` 中定义的枚举 `putimage_ops` 给出了这些算子的名字：

名 字	值	描 述
<code>COPY_PUT</code>	0	拷贝
<code>XOR_PUT</code>	1	异或
<code>OR_PUT</code>	2	或
<code>AND_PUT</code>	3	与
<code>NOT_PUT</code>	4	拷贝源图象的非

也就是说：`COPY_OUT` 将源位图象拷贝到屏幕上；`XOR_PUT` 将源图象同已在屏幕上的图象“异或”；`OR_PUT` 将源图象同屏幕上图象相“或”等等。

返 回 值 **imagesize** 返回所需存储区的大小。如果所选图象的大小大于或等于 64K 字节，**imagesize** 返回 `0xFFFF(-1)`。

getimage 和 **putimage** 无返回值

可移植性 在 Turbo Pascal 5.0 中相似子程序

GetIndex() - 获取指定信号类型的调度表下标。 -- `signal.c`

源 程 序

```
static int GetIndex(int SigType)
{
    register int i;
    for (i = 0; i < sizeof(IxGen); i++)
        if (IxGen[i] == SigType)
            return i;
    return -1;
}
```

getlinesettings 取当前线型、模式和宽度

用 法 #include <graphics.h>

void far getlinesettings(struct linesettingstype far *lineinfo);

相关函数

用 法 void far setlinestyle(int linestyle,unsigned upattern,int thickness);

说 明 getlinesettings 将有关当前的线型、模式和宽度信息存到由 lineinfo 所指的 linesettingstype 结构中。

调用 setlinestyle 可以改变这些值。该函数为所有的由 line, lineto, rectangle, drawpoly, arc, circle, ellipse, pieslice 等画的线设置线类型。

linesettingstype 结构在 GRAPHICS.H 中定义如下:

```
struct linesettingstype {
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

linestyle 说明以何种线型来画以后的线(如用实线、点线、中心线、破折线)。

在 GRAPHICS.H 中定义的枚举类型 line_styles 给出了以下线型名:

名 字	值	说明
SOLID_LINE	0	实线
DOTTED_LINE	1	点线
CENTER_LINE	2	中心线
DASHED_LINE	3	破折线
USERBIT_LINE	4	用户定义的线型

thickness 指明以后所画的线的宽度是正常的还是粗的。

名 称	值	描 述
NORM_WIDTH	1	一个像素宽
THICK_WIDTH	3	三个像素宽

upattern 是一个仅当 linestyle 是 USERBIT_LINE(4)时方起作用的 16 位模式。在这种情况下,只要模式字里有一位是 1,则线中的对应像素就用当前颜色画出来。例如,实线对应 upattern 值为 0XFFFF,破折线对应 upattern 值为 0X3333 或 0X0F0F。如果 setlinestyle 的 linestyle 参数不是 USERBIT_LINE (!=4),则 upattern 参数仍需提供,但被忽略不起作用。

返 回 值 无。

如果把无效输入传给了 setlinestyle,graphresult 返回-11,且保持当前线型不变。

可移植性 在 Turbo Pascal 5.0 中有相似的程序

getmaxcolor 返回最大的颜色值

用 法 #include <graphics.h>

int far getmaxcolor(void);

原 型 在 graphics.h

说 明 getmaxcolor 返回当前图形驱动程序和模式下最大的有效颜色值(调色板尺寸-1)。

返回值 返回最大的可用颜色值
可移植性 在 Turbo Pascal 5.0 中有相似的程序
参 见 getbkcolor, getpalette

getmaxx 返回屏幕的最大 X 坐标
用 法 #include <graphics.h>
int far getmaxx(void);

相关函数

用 法 int far getmaxy(void);
原 型 在 graphics.h

说 明 getmaxx 返回当前图形驱动程序和模式下最大的(相对于屏幕)X 值。
getmaxy 返回当前图形驱动程序和模式下最大的(相对于屏幕)Y 值。
如: 在 CGA 320X200 模式下, getmaxx 返回 319, getmaxy 返回 199。
getmaxx 和 getmaxy 对于屏幕上一个区域的中心定位和边界确定, 是非常有用的。

参 见 getx

getmaxy 返回屏幕最大的 Y 坐标
用 法 #include <graphics.h>
int far getmaxy(void);

原 型 在 graphics.h
说 明 见 getmaxx

getmodename 返回含有指定驱动程序模式名字的字符串
用 法 #include <graphics.h>
char *far getmodename(int mode_name);

原 型 在 graphics.h
说 明 getmodename 返回指定模式的名字, 其中 mode_name 为模式号。
可移植性 在 Turbo Pascal 5.0 中有相似的程序
参 见 getmaxmode, getmoderange

getmoderange 取给定图形驱动程序的模式范围
用 法 #include <graphics.h>
void far getmoderange(int graphdriver, int far *lomode,
int far *himode);

原 型 在 graphics.h
说 明 getmoderange 取得一个给定图形驱动程序 graphdriver 的有效图形模式范围。最低的允许模式值在 *lomode 中返回, 最高值在 *himode 中返回。如果 graphdriver 给出一个无效的图形驱动程序, 则 *lomode 和 *himode 均被置为 -1。

返回值 无
参 见 initgraph, getgraphmode

getpalette 返回有关当前调色板的信息
用 法 #include <graphics.h>

```
void far getpalette(struct palettetype far *palette);
```

相关函数 `int far getpalettesize(void);`

用 法 `void far setallpalette(struct palettetype far *palette);`
`void far setpalette(int index,int actual_color);`

原 型 在 `graphics.h`

说 明 `getpalette` 将有关当前调色板尺寸和颜色信息填入到由 `palette` 所指的 `palettetype` 结构中。
`getpalettesize` 返回当前图形驱动程序模式允许的调色板入口数目
`setpalette`(或 `setallpalette`)用于在 EGA/VGA 调色板中部分(或全部)改变颜色。在 CGA 中, 只能调用 `setpalette` 来改变调色板的第一个入口项(`index = 0`,背景颜色)。
`setallpalette` 把当前调色板设置为由 `palette` 所指的 `palettetype` 结构中给出的值。
`setpalette` 将调色板中的 `index` 入口项改变为 `actual_color`。例如, `setpalette(0,5)` 将当前调色板中的第一种颜色(背景颜色)改为实际颜色数 5。如果 `size` 是当前调色板的项数, 则 `index` 的范围为 0 到 `size-1`。
`palettetype` 结构(由 `getpalette` 和 `setallpalette` 所使用)和 `MAXCOLORS` 常量在 `GRAPHICS.H` 中定义如下:

```
# define MAXCOLORS 15
struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS+1];
};
```

`size` 给出当前图形驱动程序和当前模式下调色板的颜色数目。
`colors` 是一个具有 `size` 字节的数组, 它含有对应于调色板中每一个入口项实际的原始颜色编号。在 `setallpalette` 的例程中, 如果 `colors` 某一个元素是 -1, 则那一个入口项的调色板的颜色将不改变。

传给 `setpalette` 的参数 `actual_color` 和由 `setallpalette` 使用的 `colors` 数组中的元素在 `GRAPHICS.H` 中由符号常量定义如下:

实际颜色值

CGA		EGA/VGA	
名字	值	名字	值
BLACK	0	EGA_BLACK	0
BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_BROWN	20
LIGHTGRAY	7	EGA_LIGHTGRAY	7
DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61

YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

注意：有效颜色依赖于当前图形驱动程序和当前图形模式。

调色板上的变化可以立即在屏幕上看到。每当一个调色板颜色改变时，屏幕上所有出现该颜色的象素均改为新颜色。

返回值 无。

如果无效输入传给了 setpalette.graphresult 返回-11，当前调色板不变。

可移植性 在 Turbo Pascal 5.0 中有相似的程序

参 见 getbkcolor, getcolor

getpass 读一个口令

-- getpass.c

用 法 char *getpass(char *prompt);

原型在 conio.h

说 明 在用以空字符终结的字符串 prompt 提示信息后，getpass 从系统控制台读入一口令，并禁止回显。它返回一个指针，指向以空字符终结的字符串，最多为八个字符(不包括空终结符)。

返回值 返回值为指向静态字符串的指针，每次调用时都被重写。

可移植性 适用于 UNIX 系统

源 程 序

```
#include <conio.h>
#include <stdio.h>
#include <dos.h>
char *getpass(const char *prompt)
{
    register char *cp;
    register int i;
    static char xc[9];
    /* Print the prompt message */
    fprintf(stderr, "%s", prompt);
    /* Flush the keyboard buffer */
    KbdFlush();
    /* Read the password from keyboard without echo */
    for (cp = xc, i = 0; i < 8; i++, cp++)
        if ((*cp = bdos(7,0,0)) == '\r')
            break;
    *cp = '\0'; /* Password is a NULL terminated string */
    bdos(0x02, '\r', 0); /* Display a new line */
    bdos(0x02, '\n', 0);
    /* Read any remaining characters */
    KbdFlush();
    return(xc);
}
```

getpixel 取得指定象素的颜色

用 法 #include <graphics.h>

int far getpixel(int x, int y);

相关函数

用 法 void far putpixel (int x, int y);

原型在 graphics.h

说 明 getpixel 取得位于(x, y)处的象素颜色。putpixel 按 pixelcolor 定义的颜色在(x, y)处画一点

返回值 getpixel 返回给定象素的颜色值，putpixel 无返回值

可移植性 在 Turbo pascal 5.0 中有相似的程序

参 见 getimage

getpsp 取程序段前缀

-- getpsp.c

用 法 unsigned getpsp(void);

原 型 在 dos.h

说 明 getpsp 使用 dos 调用 0x62 获取程序段前缀(psp)的地址。

此调用只在 DOS 3.X 中存在。对于 2.X 和 3.X, 由启动代码设置的全局变量 _PSP 来代替。

返 回 值 getpsp 返回 PSP 的段值

可移植性 只适用于 MS-DOS 3.X, 不适用于 MS-DOS 的早期版本。

参 见 _PSP(变量)

源 程 序

```
#include <dos.h>
unsigned getpsp(void)          /* DOS 3.0 */
{
    _AH = 0x62;
    _geninterrupt(0x21);
    return(_BX);
}
```

gets 从流中取一字符串

-- gets.c

用 法 char *gets(char *string);

相关函数 char *cgets (char *string);

用 法 char *fgets (char *string, int n, FILE *stream);

原 型 在 stdio.h (fgets, gets) conio.h (cgets)

说 明 从标准输入流 stdin 中读入一字符串到 string。并用空字符(\0)代替输入中的换行符。

cgets 从控制台读入一字符串, 并把该字符串(和字符串长度)存到由 string 所指的位置中。

cgets 读字符, 直到遇上 CR/LF 组合或已读了最大允许的字符数为止。如果 cgets 读了 CR/LF 组合, 则在存储之前用空字符(\0)来代替它。

在调用 cgets 以前, 必须将要读的字符串最大长度存到 string[0] 中。返回时, string[1] 被置为实际读的字符数。实际字符串内容从 string[2] 开始, 以空字符终结。因此, string 的长必须至少为 string[0]+2 个字节。

fgets 从流 stream 中读字符存到字符串 string 中: 当读了 n-1 个字符或遇到换行符时, 函数停止读过程。fgets 保留换行字符, 读入 string 的最后一个字符后面紧跟一空字符。

返 回 值 gets 和 fgets 在调用成功时, 返回字符串参数 string; 在出错或遇到文件结束时, 返回 NULL。

cgets 返回 &string[2], 为指向所读字符串的指针, 无错误返回。

可移植性 适用于 UNIX 系统。fgets 在 Kernighan 和 Ritchie 的书中也作了定义。

参 见 ferror, fopen, fread, getc, puts, scanf

源 程 序

```
#include <stdio.h>
char *gets(char *s)
{
    register int c;
    register char *P;
    P = s;
    while ((c = getc (stdin)) != EOF && c != '\n')
        *P++ = c;
    if (EOF == c && P == s)
        return NULL;
    *P = 0;
    return ((ferror (stdin)) ? NULL : s);
}
```

getswitchchar 获取 MS-DOS 开关字符。 -- getswit.c

用 法 char getswitchchar(void);

原型文件 dos.h

说 明 返回当前 MS-DOS 开关字符值。

返回 值 当前 MS-DOS 开关字符值。

源 程 序

```
#include <dos.h>
int getswitchchar(void)
{
    _AX = 0x3700;
    _geninterrupt(0x21);
    return(_DL);
}
```

gettext 将文本方式屏幕上的文本拷贝到存储区 -- gptext.c

用 法 int gettext (int left, int top, int right, int bottom,
void *destin);

相关函数

用 法 int puttext (int left, int top, int right, int bottom,
void *source);

原 型 在 conio.h

说 明 gettext 把屏幕上由 left、top、right 和 bottom 定义的矩形区域的内容存入由 destin 所指的存储区域。

puttext 将 source 所指的存储区域中的内容写到由 left、top、right 和 bottom 所定义的屏幕区域中。

所有的坐标都为绝对屏幕坐标，而不是相对于窗口的坐标。

gettext 将矩形中的内容从左到右、自上而下顺序读入到存储区中。puttext 用同样的方式将存储区中的内容放到所定义的矩形中。

屏幕上的每个位置占两个字节内存：第一字节是单元中的字符，第二字节是单元的屏幕显示属性。一个长为 h 行宽为 w 列的矩形所需的空间定义为：

字节数(bytes)=(h*w)*2

返回 值 如果操作成功，这些函数返回 1；如果失败(如给出的坐标超过当前屏幕模式范围)，返回 0。

可移植性 只适用于 IBM PC 及与 BIOS 兼容的系统

参 见 movetext

源 程 序

```
#include <video.h>
#include <conio.h>
int gettext(int left, int top, int right, int bottom, void *buffer)
{
    int y, size;
    if (!__validatexy(left, top, right, bottom))
        return 0;
    size = right-left+1;
    for (y = top; y <= bottom; y++)
    {
        _screenio(buffer, __vptr(left, y), size);
        (char *)buffer += size*2;
    }
    return 1;
}
```

gettextinfo 取得文本模式的显示信息 -- vidinfo.c

用 法 #include <conio.h>

```
void gettextinfo(struct text_info *infoec);
```

原型在 conio.h

说明 gettextinfo 将当前文本显示信息填入 infoec 所指的 text_info 结构中。

text_info 结构在 CONIO.H 中定义如下:

```
struct text_info {  
    unsigned char winleft; /* left window coordinate */  
    unsigned char wintop; /* top window coordinate */  
    unsigned char winright; /* right window coordinate */  
    unsigned char winbottom; /* bottom window coordinate */  
    unsigned char attributes; /* text attribute */  
    unsigned char normattr; /* normal attribute */  
    unsigned char curmode; /* BW40, BW80, C40, OR, C80 */  
    unsigned char screenheight; /* bottom_top */  
    unsigned char screenwidth; /* right_left */  
    unsigned char curx; /* X coordinate in current window */  
    unsigned char cury; /* Y coordinate in current window */  
};
```

返回值 无。结果在 infoec 所指结构中返回

可移植性 只适用于 IBM PC 及其兼容机

参见 textattr, textbackground, textcolor, textmode, wherex, wherey, window

源程序

```
#include <_video.h>  
#include <conio.h>  
void gettextinfo(struct text_info *r)  
{  
    *r = *(struct text_info *)& _video; /* move data all at once */  
    r->winleft += 1; /* 1,1 origin */  
    r->wintop += 1;  
    r->winright += 1;  
    r->winbottom += 1;  
    r->curx = wherex();  
    r->cury = wherey();  
}
```

gettextsettings 返回有关当前文本设置的信息

用法 #include <graphics.h>

```
void far gettextsettings(struct textsettingstype far *textinfo);
```

相关函数 void far settextjustify (int horiz, int vert);

用法 void far settextstyle (int font, int direction, int charsize);

原型在 graphics.h

说明 将有关当前字体、方向、大小和对齐方式(由 settextstyle 和 settextjustify 设置)的信息填入由 textinfo 所指的 textsettingstype 结构中。

gettextsettings 所用到的 textsettings 结构在 GRAPHICS.H 中定义如下:

```
struct textsettings {  
    int font;  
    int direction;  
    int charsize;  
    int horiz;  
    int vert;  
};
```

调用 `settextjustify` 之后的文本输出将按指定模式在水平和垂直方向上与 `cp` 对齐。缺省的对齐模式是 `LEFT_TEXT`(水平的)和 `TOP_TEXT`(垂直的)。 `GRAPHICS.H` 中的枚举类型 `text_just` 提供传送给 `horiz` 和 `vert` 的符号名:

名字	值	说明
<code>LEFT_TEXT</code>	0	水平
<code>CENTER_TEXT</code>	1	水平和垂直
<code>RIGHT_TEXT</code>	2	水平
<code>BOTTOM_TEXT</code>	0	垂直
<code>TOP_TEXT</code>	2	垂直

如果 `horiz` 等于 `LEFT_TEXT` 并且 `direction=HORIZ_DIR`, 则 `cp` 的 `x` 分量(`cpx`) 在调用 `outtext(string)` 之后将向前移动 `textwidth(string)`。

`settextstyle` 设置文本字体、文本显示方向和字符的大小。对 `settextstyle` 的调用将影响所有由 `outtext` 和 `outtextxy` 产生的字符输出。

传给 `settextstyle` 的参数 `font`, `direction` 和 `charsize` 描述如下:

`font`: 有一个 `8X8` 位图字体和几个“矢量”字体可用。缺省为 `8X8` 位图字体。在 `GRAPHICS.H` 中定义的枚举类型 `font_name` 为不同的字体提供了符号名:

名字	值	说明
<code>DEFAULT_FONT</code>	0	位图字体
<code>TRIPLEX_FONT</code>	1	三重矢量字体
<code>SMALL_FONT</code>	2	小号矢量字体
<code>SANSERIF_FONT</code>	3	无衬线矢量字体
<code>GOTHIC_FONT</code>	4	哥特矢量字体

缺省的位图字体建立在图形系统中, 矢量字体则存放在 `*.CHR` 磁盘文件上, 且每次只有一个文件被保存在存储区中。这样, 当用户选择一种键入的字体时(与上次选择的键入字体不同), 相应的 `*.CHR` 文件须从键盘装入。为了在使用几个矢量字体时避免这种装入, 可以将字体文件连接到程序中。为此, 可以用 `BGIOBJ` 应用程序将它们转换为目标文件, 然后通过 `registerbgifont` 把它们注册, 然后进行连接。

`direction`: 所支持的字体方向是水平文本(从左到右)和垂直文本(顺时针旋转), 缺省的方向是 `HORIZ_DIR`:

名	值	描述
<code>HORIZ_DIR</code>	0	从左到右
<code>VERT_DIR</code>	1	自底向上

`charsize`: 用 `charsize` 因子可以将每个字符的大小放大。如果 `charsize` 非零, 它可以影响位图或矢量字符; 如果 `charsize` 为零, 只影响矢量字符。

- 如果 `charsize=1`, `outtext` 和 `outtextxy` 将把 `8X8` 位图字体的字符在屏幕上显示为 `8X8` 的象素矩阵。
- 如果 `charsize=2`, 这些输出函数将把 `8X8` 图形字体的字符显示为 `16X16` 的象素矩阵, 依次类推(最大到普通大小的 10 倍)

- * 当 `charsize=0` 时, 输出函数 `outtext` 和 `outtextxy` 用缺省的字符放大因子(4)或者用 `setusercharsize` 给出的用户定义字符大小来放大矢量字体。

要用 `textheight` 和 `textwidth` 来确定字符的实际大小。

返回值 无。

由于矢量字体可以存储在磁盘上, 所以装入时可能会发生错误。如果发生了错误, `graphresult` 将返回下列值之一:

- 8 未找到字体文件
- 9 没有足够的存储区域装入所选的字体
- 11 一般错误
- 12 图形I/O错误
- 13 无效字体文件
- 14 无效字体号

如果传给 `settextjustify` 的是无效的输入, `graphresult` 将返回-1, 而当前文本对齐方式不变。

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 `graphresult`, `outtext`, `registerbgifont`, `textheight`

gettime 取得系统时间 -- getdate.c

用 法 `#include <dos.h>`

`void gettime (struct time *timep);`

原 型 在 `dos.h`

说 明 见 `getdate`

源 程 序

```
#include <dos.h>
void gettime(struct time *timep)
{
    _AH = 0x2c;
    _geninterrupt(0x21);
    ((int *)timep)[0] = _CX;
    ((int *)timep)[1] = _DX;
}
```

getvect 取得中断向量入口 -- getvect.cas

用 法 `void interrupt(*getvect (int intr_num))();`

相关函数

用 法 `void setvect(int intr_num void, interrupt(*isr)());`

原 型 在 `dos.h`

说 明 MS-DOS 包括一系列“硬线”从 0 号到 255 号的中断向量, 每个向量的 4 字节值实际上是中断处理函数的入口地址。

`getvect` 读名为 `intr_num` 的向量值, 并把该值解释为中断函数的(远)地址。

`setvect` 把名为 `intr_num` 的向量值置为新值 `vector`, `vector` 是包含新中断函数地址的远指针。

如果 C 子程序定义为 `interrupt` 例程, 则该例程的地址只能传送给 `vector`。

注意: 如果使用了 `dos.h` 中所定义的原型, 则可简单地把中断函数地址传送给任何存储模式下的 `setvect`。

返回值 `getvect` 返回存在 `intr_num` 中断向量中的当前 4 字节值。`setvect` 无返回值。

可移植性 只适用于 MS-DOS

参 见 `disable`

源 程 序

```

#pragma inline
#include <dos.h>
void interrupt (far* getvect(int intr))() {
    asm    mov     ah, 035h
    asm    mov     al, intr
    asm    int     021h
#pragma warn -sus
    return (char _es *) _BX;
#pragma warn .sus
}

```

getverify 取得验证状态 -- getverl.cas

用 法 int getverify(void);

相关函数

用 法 void setverify (int value);

原型在 dos.h

说 明 getverify 取得验证标志的当前状态。

setverify 设置当前验证标志的状态为 value。

value 为 0=验证标志 off

value 为 1=验证标志 on

验证标志控制磁盘输出。当其为 off 时，输出不验证；当其为 on 时，所有磁盘写都被验证，以保证数据被正确写入。

返回 值 getverify 返回验证位的当前状态 0 或 1

返回 0=验证标志 off

返回 1=验证标志 on

setverify 无返回值

可移植性 只适用于 MS-DOS

源 程 序

```

#pragma inline
#include <dos.h>
int getverify(void) {
    asm    mov     ax, 054001h
    asm    xor     dx,dx
    asm    int     021h
    asm    xor     ah,ah
    return _AX;
}

```

getviewsetting 返回有关当前视区的信息

用 法 #include <graphics.h>

void far getviewsettings(struct viewporttype far *viewport);

相关函数 void far setviewport(int left, int top, int right,

int bottom, int clipflag);

原型在 graphics.h

说 明 getviewsettings 将有关当前视区的信息填入到 viewport 所指的 viewporttype 结构中。

setviewport 为图形输出建立一个新的视区。

视区的两角用绝对屏幕坐标(left, top)和 (right, bottom)给定。当前位置(cp)移到视区的(0, 0)。

参数 clipflag 决定画线是否在当前视区的边界处剪切掉(截断)。当程序调用 setviewport 时，如果 clipflag 为非零值，则所有的图形在当前视区边缘区被剪切掉。

getviewport 使用的 viewporttype 结构在 GRAPHICS.H 中定义如下:

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

注意: initgraph 和 setgraphmode 将视区复位为整个图形屏幕。

返回值 无

如果无效输入传给了 setviewport, 则 graphresult 返回-1, 当前的视区设置保持不变。

可移植性 在 Turbo Pascal 5.0 中有相似的程序。

getw 从流中取一整数

-- getw.c

用法 #include <stdio.h>

```
int getw(FILE *stream);
```

原型 在 stdio.h

说明 见 getc

源程序

```
#include <stdio.h>  
int getw(FILE *fp)  
{  
    int c, res;  
    if ((c = getc(fp)) != EOF)  
        if ((res = getc(fp)) == EOF)  
            c = EOF;  
    else  
        (*((char *)&c + 1)) = res;  
    return(c);  
}
```

getx 返回当前位置的 X 坐标

用法 #include <graphics.h>

```
int far getx(void);
```

相关函数

用法 int far gety (void);

原型 在 graphics.h

说明 getx 返回当前位置的 X 坐标。

gety 返回当前位置的 Y 坐标。

返回值 getx 返回 cp 的 X 坐标, gety 返回 cp 的 Y 坐标(坐标值是相对于视区的)。

可移植性 在 Turbo Pascal 5.0 中有相似的程序

参见 getviewsettings, initgraph, moveto

gety 返回当前位置的 Y 坐标

用法 #include <graphics.h>

```
int far gety(void);
```

原型 在 graphics.h

说明 见 getx

gmtime 把日期和时间转换为格林威治标准时间

--- ctime.c

用法 #include <time.h>

struct tm *gmtime(long *clock);

原型在 time.h

说明 见 ctime

源程序

```
#include <io.h>
#include <time.h>
#include <stdio.h>
static const char Days[12] = {
    31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};
static const char *const Weekday[7] = {
    "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};
static const char *const Months[12] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
static struct tm tm;
struct tm *gmtime(const long *clock)
{
    return(comtime(*clock, 0));
}
```

gotoxy 在文本窗口中设置光标

-- gotoxy.c

用法 void gotoxy (int x, int y);

原型在 conio.h

说明 gotoxy 在当前文本窗口中移动光标到指定的位置。如果坐标无效，则对 gotoxy 的调用被忽略，如当窗口右下角的位置是(35, 25)时，调用 gotoxy(40, 30)。

wherex 和 wherey 函数分别返回光标的当前 x 和 y 位置。返回值 无

可移植性 只适用于 IBM PC 及其兼容机，在 Turbo Pascal 5.0 中有相似的程序。

参见 wherex, wherey, window

源程序

```
#include <video.h>
#include <conio.h>
void gotoxy(int column, int row)
{
    byte r,c;
    /* translate to physical coord */
    r = row-1; r += _video.windowy1;
    c = column-1; c += _video.windowx1;
    if ((r < _video.windowy1) || (r > _video.windowy2) || (c < _video.windowx1) || (c > _video.windowx2))
        return;
    _DL = c;
    _DH = r;
    _AH = V SET_CURSOR_POS;
    _BH = 0;
    _VideoInt();
} /* gotoxy */
```

graphdefaults 将所有图形设置重置为它们的缺省值

用法 #include <graphics.h>

void far graphdefaults (void);

原型在 graphics.h

说明 graphdefault 将所有图形设置重置为它们的缺省值。即：

- 将视区置为整个屏幕
- 将当前位置移到(0, 0)
- 设置缺省的调色板颜色、背景颜色和画线颜色

- 设置缺省的填充类型和模式
- 设置缺省的文本字体和对齐方式

返回值 无

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 initgraph, getgraphmode

grapherrormsg 返回一个错误信息字符串

用 法 #include <graphics.h>

char *far grapherrormsg(int errorcode);

原 型 在 graphics.h

说 明 见 graphresult

_graphexit 终止程序(用于图形库)。 -- gexit.c

用 法 void far _graphexit(int status);

原型文件 graphics.h

说 明 这是 graphexit 的缺省版本。它很少用指定的退出状态来调用 exit。

源 程 序

```
#include <graphics.h>
#include <process.h>
void far _graphexit(int status)
{
    exit(status);
}
```

_graphfreemem 用户可修改的图形存储区释放函数 -- gfreemem.c

用 法 #include <graphics.h>

void far _graphfreemem(void far *ptr, unsigned size);

原 型 在 graphics.h

说 明 见 _graphgetmem

源 程 序

```
#include <graphics.h>
#include <alloc.h>
void far _graphfreemem(void far *ptr, unsigned size)
{
    #pragma warn -sus
    free(ptr);
    #pragma warn .sus
    #pragma warn -par
}
#pragma warn .par
```

_graphgetmem 用户可修改的图形存储区分配函数 -- ggetmem.c

用 法 #include <graphics.h>

void far *far _graphgetmem(unsigned size);

相关函数

用 法 void far _graphfreemem(void far *ptr, unsigned size);

原 型 在 graphics.h

说 明 图形库调用 _graphgetmem 为内部缓冲区、图形驱动程序和字符集分配存储区。

通过定义自己的 _graphgetmem(必须准确地按所示用法定义), 用户可以控制图形库的内存管理。该例程序的缺省版本只是调用 malloc。

图形库调用 `_graphfreemem` 释放 `_graphgetmem` 所分配的内存。通过定义自己的 `_graphfreemem` (必须准确地按所示用法定义), 用户可以控制图形库的内存管理。该例程序的缺省版本只是调用 `free`。

返回值 无

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

源程序

```
#include <graphics.h>
#include <alloc.h>
void far * far _graphgetmem(unsigned size)
{
    return(malloc(size));
}
```

graphresult 返回最后一次不成功的图形操作的错误代码

用法 `#include <graphics.h>`
`int far graphresult(void);`

相关函数

用法 `char *far grapherrormsg(int errorcode);`

原型在 `graphics.h`

说明 **graphresult** 返回最后一次出错图形操作的错误代码。

grapherrormsg 返回同 `errorcode` 相联系的字符串的指针, 而 `errorcode` 是 **graphresult** 返回的错误代码。 **grapherrormsg** 使得程序可以很容易地显示一个描述性错误信息。如用 “Device driver not found (CGA,BGI)” 代替 “error code-3”, 这样使程序易读。下表列出 **graphresult** 返回的错误代码, 与错误代码相联系的 `graphics_errors` 类型和相应的错误信息。

错误代码	<code>graphics_errors</code> 常量	相应的错误信息字符串
0	<code>grOk</code>	No error
-1	<code>grNotInitGraph</code>	(BGI)graphics not installed(use <code>initgraph</code>)
-2	<code>grNotDetected</code>	Graphics hardware not detected
-3	<code>grFileNotFound</code>	Device driver file not found
-4	<code>grInvalidDriver</code>	Invalid device driver file
-5	<code>grNoLoadMem</code>	Not enough to load driver
-6	<code>grNoScanMem</code>	Out of memory in scan fill
-7	<code>grNoFloodMem</code>	Out of memory in flood fill
-8	<code>grFontNotFound</code>	Font file not found
-9	<code>grNoFontMem</code>	Not enough memory to load font
-10	<code>grInvalidMode</code>	Invalid graphics mode for selected driver
-11	<code>grError</code>	Graphics error
-12	<code>grIOError</code>	Graphics I/O error
-13	<code>grInvalidFont</code>	Invalid font file
-14	<code>grInvalidFontNum</code>	Invalid font number
-15	<code>grInvalidDeviceNum</code>	Invalid device number
-18	<code>grInvalidVersion</code>	Invalid graphics version

下列函数影响graphresult返回值

bar	bar3d	clearviewport
closegraph	detectgraph	drawpoly
fillpoly	floodfill	getgraphmode
imagesize	initgraph	installuserdriver
installuserfont	pieslice	registerbgidriver
registerbgifont	setallpalette	setfillpattern
setfillstyle	setgraphbufsize	setgraphmode
setlinestyle	setpalette	settextjustify
settextstyle		

注意: graphresult 被调用之后, 内部错误返回代码复位为零, 因此应该把 graphresult 的返回值存进临时变量中, 然后再测试该变量。

返回值 graphresult 返回当前图形错误编号, 即一个范围从-18 到 0 的整数。

grapherrormsg 返回一个指向 graphresult 返回值相联系的串的指针。

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 initgraph

harderr 建立一个硬件错误处理程序 -- harderr.cas

用 法 void harderr (int (*fptr)());

相关函数 void hardresume (int rescode);

用 法 void hardretn(int errcode);

原 型 在 dos.h

说 明 harderr 为当前程序建立一个硬件错误处理程序。每当发生 0X24 中断时, 激活该处理程序(参阅《MS-DOS 程序员参考手册》中有关中断的讨论)。当此中断发生时, 调用 fptr 所指的函数。该处理函数接受下列参数:

handler(int errval, int ax, int bp, int si)

其中: errval 是 MS-DOS 置于 DI 寄存器的出错代码, 而 ax, bp 和 si 分别是 MS-DOS 置于 AX, BP 和 SI 寄存器的值。

- * ax 表示是否遇到了磁盘错误或其它设备错误。如果 ax 为非负值, 表示遇到了磁盘错误, 否则为设备错误。对于磁盘错误, ax 的值“与”上 0X00FF 后, 给出出错驱动器号, (1=A, 2=B 等等)。

- * bp 和 si 一起指向出错驱动器的设备驱动程序头。

给定名函数不是直接被调用, harderr 建立一个 DOS 中断处理程序来调用它。

peek 和 peekb 用于从驱动程序头检索设备的信息。bp 为段地址, si 为偏移量。

处理程序可以通过 0XC 发出 bdos 号调用, 但任何其它的 bdos 调用将使 MS-DOS 崩溃。特别是不能使用 C 标准 I/O 和 UNIX 仿真 I/O。

驱动程序头不可以用 poke 或 pokeb 改变。

错误处理程序可以返回或通过调用 hardresume 返回到 MS-DOS。处理程序的

返回值或 hardresume 的 rescode(结果代码)包含 abort(2)、retry(1)或 ignore(0)指示标志。

abort 是通过 DOS 中断 0X23 实现的, 即 Control_Break 中断。

错误处理程序也可以通过调用 hardretn 直接返回到应用程序。

返回值 处理程序返回 0(对 ignore), 1(对 retry)或 2(对 abort)。

可移植性 只适用于 MS-DOS

参 见 peek, poke, setjmp

源 程 序

```
#pragma inline
#include <dos.h>
static int Hsav;
static int (*Hfunc)(unsigned di, unsigned ax, unsigned bp, unsigned si);
void harderr(int (*fptr)())
{
    Hfunc = fptr;
    setvect(0x24, hentry);
}
```

hardresume 硬件错误处理函数

-- harderr.cas

用 法 void hardresume(int rescode);

原 型 在 dos.h

说 明 见 harderr

源 程 序

```
void hardresume(int axret)
{
    AX = axret;
    SP = Hsav;
asm    pop    bp
asm    pop    di
asm    pop    si
asm    pop    ds
asm    pop    es
asm    pop    dx
asm    pop    cx
asm    pop    bx
asm    inc    sp;          /* Don't restore ax */
asm    inc    sp;
asm    iret
}
```

hardretn 硬件错误处理函数

-- harderr.cas

用 法 void hardretn(int errcode);

原 型 在 dos.h

说 明 见 harderr

源 程 序

```
void hardretn(int retn)
{
    hdos(0x54,0,0);          /* Clean up DOS */
    AX = retn;
    BP = SP = Hsav + 26; /* Restore stack and make it addressable */
asm    or     byte ptr 20[bp], 1 /* Set carry flag */
asm    pop    bx
asm    pop    cx
asm    pop    dx
asm    pop    si
asm    pop    di
asm    pop    bp
asm    pop    ds
asm    pop    es
asm    iret          /* Hop back to user */
}
```

_heaplen 堆栈长度。

-- heaplen.c

用 法 extern unsigned _heaplen;

说 明 该模块定义缺省的近堆栈大小。

源 程 序

```
#include <asmrules.h>
#if !(LDATA)
unsigned      _heaplen = 0x0000;      /* To expand DS as much as possible */
#endif
```

hentry 调用用户 Ctrl-break 处理程序的句柄。-- ctrlbrk.c

用 法 static void interrupt far hentry();

原型文件 局限于本模块

说 明 解释 Ctrl-break 处理程序的返回值。

返 回 值 无

源 程 序

```
#include <dos.h>
#include <process.h>
static int (*Hfunc)(void);
static void interrupt far hentry()
{ if (!(*Hfunc)())
    _exit(0);
}
```

函 数 名 Hex4 - 将整数转换为四位十六进制数。 -- vprinter.cas

用 法 static void near pascal Hex4 (unsigned datum)

说 明 将 16 位参数转换为四位十六进制数并存放在 ES:[ID]。

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <rules.h>
#include <printf.h>
#define I asm #define BREAKPOINT asm int 3
/* By defining FASTWAY, about a 1-2% literal transcription speed gain can be
obtained. The space cost of this is between 30-40 bytes however. For the
default configuration this gain was not deemed worth the space penalty. */
#undef FASTWAY
static char NullString[] = "(null)";
static char hexDigits[16] =
{ '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F',
};
static void near pascal Hex4 (unsigned datum) {
I mov dx, datum
I mov cx, 0F04h
I mov bx, offset hexDigits
I cld
I mov al, dh
I shr al, cl
I xlat
I stosb
I mov al, dh
I and al, ch
I xlat
I stosb
I mov al, dl
I shr al, cl
I xlat
I stosb
I mov al, dl
I and al, ch
I xlat
I stosb
I return;
}
```

highvideo 选择高度文本字符

-- color.c

用 法 `void highvideo(void);`

相关函数 `void lowvideo(void);`

用 法 `void normvideo(void);`

原 型 在 `conio.h`

说 明 **highvideo** 通过设置当前所选择的前景颜色的高亮度位来选择高亮度字符。
normvideo 通过将文本属性(前景和背景)置为启动程序时它具有的值来选择标准字符。
lowvideo 通过清除当前所选择的前景颜色高度位来选择低亮度字符。

返 回 值 无

可移植性 只适用于 IBM PC 及其兼容机。在 Turbo Pascal 5.0 中有相似的程序。

参 见 `cprintf`, `cpurs`, `gettextinfo`, `putch`, `textattr`

源 程 序

```
#include <_video.h>
#include <conio.h>
#define INTENSE 0x08
void highvideo(void)
{ video.attribute |= INTENSE;
}
```

hyperb 双曲函数

相关函数 `double sinh(double x);`

用 法 `double cosh(double x);`
`double tanh(double x);`

原 型 在 `math.h`

说 明 这些函数计算给定实参的双曲函数值。

返 回 值 这些函数分别返回各自的计算结果当正确值溢出时, `sinh` 和 `cosh` 返回带适当符号的 `HUGE_VAL` 值。

这些函数的错误处理程序可以通过 `matherr` 函数来修改。

可移植性 适用于 UNIX 系统

参 见 `exp`

hypot 计算直角三角形的斜边长

-- hypot.cas

用 法 `double hypot (double x, double y);`

原 型 在 `math.h`

说 明 **hypot** 计算 Z 的值, 其中 $Z = \sqrt{X^2 + Y^2}$ (若直角三角形的两直角边分别为 X , Y , 则 Z 为斜边长)

返 回 值 成功时, **hypot** 返回双精度值 Z 。在出错时(如溢出), 返回 `HUGE_VAL`, 并置 `error` 为:

`ERANGE` 结果超出范围

hypot 的错误处理程序可以通过 `matherr` 函数来修改。

可移植性 适用于 UNIX 系统

参 见 `trig`

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
```

```

#include <errno.h>
#pragma warn -rvl
double hypot (double x, double y) {
asm    FLD    DOUBLE (x)
asm    mov    ax, x [6]
asm    FMUL   st, st          /*      (z.x)^2 */
asm    shl    ax, 1
asm    cmp    ax, 0FFE0h
asm    jnb    hyp_infiniteX
asm    FLD    DOUBLE (y)
asm    mov    ax, y [6]
asm    FMUL   st, st          /*      (z.y)^2 */
asm    shl    ax, 1
asm    cmp    ax, 0FFE0h
asm    jnb    hyp_infiniteY
asm    FADD                    /*      ... + ... */
asm    FSQRT                    /* sqrt (      ) */
hyp_end:
    return;
hyp_infiniteY:
asm    FSTP   st(0)            /* pop y off stack */
hyp_infiniteX:
asm    FSTP   st(0)            /* pop x off stack */
#pragma warn -ret
    return _matherr (OVERFLOW, "hypot", &x, &y, HUGE_VAL);
#pragma warn .ret
}
#pragma warn .rvl

```

imagesize 返回保存位图象所需的字节数

用 法 #include <graphics.h>

unsigned far imagesize(int left, int top, int right, int bottom);

原 型 在 graphics.h

说 明 见 getimage

initgraph 初始化图形系统

用 法 #include <graphics.h>

void far initgraph(int far *graphdriver, int far *graphmode,
char far *pathtodriver);

相关函数 void far detectgraph(int far *graphdriver, int far *graphmode);

用 法 void far closegraph(void);

说 明 initgraph 通过从磁盘上装入一个图形驱动程序(或确认一个已注册的驱动程序)来初始化图形系统, 并将系统置为图形模式。

detectgraph 检测系统的图形适配器, 选择该适配器所能提供的最高分辨率的模式。如果没有

检测到图形硬件, 则*graphdriver 参数被置为-2, 且 graphresult 也返回-2。

注意: 直接调用 detectgraph 的主要原因是覆盖由 detectgraph 提供给 initgraph 的图形模式。closegraph 释放所有图形系统分配的存储区, 然后将屏幕恢复为调用 initgraph 之前的模式。(图形系统通过调用 _graphfreemem 来释放存储区: 如驱动程序、字体和内部缓冲区所占的存储区)

为了启动图形系统, 首先要调用 initgraph 函数。initgraph 装入图形驱动程序, 并将系统置为图形模式。initgraph 可以使用由用户指定的一个特殊的图形驱动程序和模式, 或在运行时自动检测与之关联的显示适配器, 选择一相应的驱动程序。如果用户告诉 initgraph 进行自动检测, 它就调用 detectgraph 来选择一个图形驱动程序和模式。initgraph 还将所有的图形设置(当前位置、调色板、颜色、视区等)复位为它们的缺省值, 并置 graphresult 为 0。

一般情况下, initgraph 为图形驱动程序的装入分配存储区(用 _graphgetmem), 然后从磁盘装入适当的.BGI 文件。除了这种动态装入机制外, 也可以将一个图形驱动程序文件(或几个)直接同可执行程序文件连接起来。(参看附录中有关 BGI OBJ 的说明)。

pathtodriver 指定 initgraph 寻找图形驱动程序的目录路径。initgraph 首先在 pathtodriver 指明的路径中寻找, 然后(若不在那里)在当前目录中寻找。因此, 如果 pathtodriver 为空(NULL), 则驱动程序文件(.BGI)必定在当前目录中。Settextstyle 搜索矢量字体(*.CHR)也是这个路径。

* graphdriver 是一个整数, 它指定要所用的图形驱动程序。用户可以用 graphics drivers 枚举类型中的常量对它赋值。常量在 GRAPHICS.H 中定义, 列表如下:

graphics_drivers	
常量	数值
DETECT	0 (需自动检测)
CGA	1
MCGA	2
EGA	3
EAG64	4
EGAMONO	5
IBM 8514	6
HEROMONO	7
ATT400	8
VGA	9
PC 3270	10

* graphmod 是一个整数, 它说明初始图形模式(除非*graphmode=DETECT, 在这样情况下,

*graphmode 被设置为检测到的驱动程序可用的最高分辨率)。用户可以使用 graphics_modes 枚举类型中的常量对它赋值。常量在 GRAPHICS.H 中定义, 列表如下:

图形驱动器	graphics_modes	值	列X行	调色板	页数
CGA	CGACO	0	320X320	C0	1
	CGAC1	1	320X320	C1	1
	CGAC2	2	320X320	C2	1
	CGAC3	3	320X320	C3	1
	CGAHI	4	640X200	2色	1
MCGA	MCGAC0	0	320X200	C0	1

	MCGAC1	1	320X200	C1	1
	MCGAC2	2	320X200	C2	1
	MCGAC3	3	320X200	C3	1
	MCGAMED	4	640X200	2色	1
	MCGAHI	5	640X480	2色	1
EGA	EGAL0	0	640X640	16色	4
	EGAHI	1	640X350	16色	2
EGA64	EGA64L0	0	640X200	16色	1
	EGA64HI	1	640X350	4色	1
EGAMONO	EGAMONOH1	3	640X350	2色	1(2)*
HERC	HERCMONOH1	0	720X348	2色	2
ATT400	ATT400C0	0	320X200	C0	1
	ATT400C1	1	320X320	C1	1
	ATT400C2	2	320X320	C2	1
	ATT400C3	3	320X320	C3	1
	ATT400MED	4	640X200	2色	1
	ATT400HI	5	640X640	2色	1
VGA	VGA10	0	640X400	16色	2
	VGAMED	1	640X350	16色	2
	VGAHI	2	640X480	16色	1
PC3270	PC3270HI	0	720X350	2色	1
IBM8514	IBM8514L0	0	640X480	256色	
	IBM8514HI	1	1024X768	256色	

* 卡上 64K 为 1 页, 256K 为 2 页。

上表中, 调色板列表一列里的 C0, C1, C2 和 C3 指在 CGA(或兼容卡)系统中有效的四种预定义的四色调色板。在每个调色板中, 可以选择背景颜色(入口项为#0), 但其它颜色固定不变。下表列出调色板各入口项颜色值:

调色板号	分配给象素值的颜色		
	1	2	3
0	淡绿	淡红	黄色
1	淡青	粉红	白色
2	绿色	红色	棕色
3	青色	品红色	淡灰色

调用 `initgraph` 之后, `*graphdriver` 被设置为当前图形驱动程序, `graphmode` 为当前图形模式。

返回值 无。

`initgraph` 总是要设置内部错误代码, 如果成功, 置代码为 0; 如果发生了错误, `graphdriver` 被置为 -2, -3, -4 或 -5, `graphresult` 返回同样的值, 如下所示:

- 2 不能检测图形卡
- 3 不能找到驱动程序文件

- 4 无效驱动程序
- 5 无足够存储区来加载驱动程序

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 getgraphmode, _graphgetmem, registerbgidriver, restorecrtmode, setgraphbufsize

inp 从硬件端口中输入

用 法 int inp(int portid);

原 型 在 dos.h

说 明 见 inport

inport 从硬件端口中输入

-- inport.cas

用 法 #include <dos.h>;

int inport(int portid);

相关函数 int inp(int portid);

用 法 int inportb(int portid);

void outp(int portid, int value);

void outputport(int portid, int value);

void outputportb(int portid, unsigned char value);

原 型 在 dos.h

说 明 inport 从指定端口 portid 读入一字。

inp 是一等价于 inport 的宏。

inportb 是从指定端口 portid 读入一字节的宏。

outp 是一等价于 outputport 的宏。

outputport 写给定的字值 value 到指定端口 portid。

outputb 是写一字节值 value 到指定端口 portid 的宏。

如果在调用 inportb 或 outputb 时包含了 dos.h, 则它们当作宏看待, 并扩展为插入代码。

如果没有包含 dos.h 文件, 或虽然包含了 dos.h 但使用了 #undef inportb 或 outputb 指令, 但它们被当作函数看待。

返 回 值 inp, inport, inportb 返回读入的值

outp, outputport, outputb 无返回值。

可移植性 只适用于 8086 系列源程序

```
#pragma inline
#include <dos.h>
#undef inportb
int inport(int port) {
asm    mov    dx,port
asm    in     ax,dx
    return _AX;
}
unsigned char inportb(int port) {
asm    mov    dx,port
asm    in     al,dx
asm    xor    ah,ah
    return _AX;
}
```

insline 在文本窗口中插入一个空行

-- insline.c

用 法 void insline(void);

原 型 在 conin.h

说 明 `insline` 用当前文本背景颜色, 在文本窗口的光标位置处插入一空行。空行下面的所有各行都下移一行, 底行滚出窗口底部。

返回值 无

可移植性 只适用在 IBM PC 及其兼容机上, 在 Turbo pascal 中有相似的子程序

参 见 `clreol`, `delline`, `window`

源 程 序

```
#include < video.h>
#include < conio.h>
void insline(void)
/* Inserts a line at the current line */
{
    _scroll(DOWN, _video.windowx1, wherey(),
            _video.windowx2, _video.windowy2, 1);
} /* insline */
```

installuserdriver 安装用户设备驱动程序到 BGI 设备驱动程序表中。

用 法 `#include <graphics.h>`

`int far installuserdriver(char far *name, int (*detect)(void));`

原型在 `graphics.h`

说 明 函数用于安装一个厂商提供的设备驱动程序。name 是新设备驱动程序的名字(x.BGI), detect 为一自动检测初始化函数, 它没有参数, 返回一指向整型的指针。

有两种使用厂商提供的驱动程序的方法。假设 SGA 新显示卡加工厂提供 SGA.BGI 设备驱动程序, 最简单的使用方法是调用 `installuserdriver` 安装, 直接把返回的安装号传给 `initgraph`:

```
int graphics, graphmode;
graphdriver = installuserdriver("SGA.BGI", NULL);
graphmode = 0;
initgraph(&graph driver, &graphmode, "");
```

另一种是由厂商提供自动检测函数 `detectSGA()`:

```
graphdriver = installuserdriver("SGA.BGI", detectSGA());
initgraph(&graphdriver, &graphmode, "");
```

其中 `graphmode` 由 `detectSGA()` 返回。

返回值 成功时, 本函数返回设备驱动程序的安装号; 如果出错, `graphresult` 返回下列值之一:

- 3 无此文件
- 4 无效驱动程序名
- 5 内部驱动程序表满
- 11 一般错误

参 见 `installuserfont`

installuserfont 安装用户提供的字体文件

用 法 `#include <graphics.h>`

`int far installuserfont(char far *name);`

原型在 `graphics.h`

说 明 本函数用于安装一新的字体文件, name 为 dos 下的新字体文件名(*.CHR)。

本函数返回一 ID 号, 为使用新字体可调用 `settextstyle` 函数。

返回值 安装成功时, 返回新字体安装号; 否则 `graphresult` 返回下列值之一:

- 8 未找到字体文件
- 9 内部字体表满
- 11 一般错误
- 12 图形I/O错误
- 13 无效的字体文件
- 14 无效的字体号

参 见 `settextstyle`

Int0Catcher() - 除零处理程序 (该过程是对包括整数算术运算例外处理 SIGFPE 的扩充的一部分。ANSI 并不限制 SIGFPE 只用于浮点数)。

-- signal.c

说 明 每当用 SIGFPE 调用 `signal` 时就安装该程序。激活时该过程与 `raise(SIGFPE, ...)` 等价。对于以后程序, 该程序一直有效。如果应用程序不调用 SIGFPE 处理程序, 则使用由 C 启动代码设置的缺省动作。如果应用程序借用了 INT0, 则只能用 `signal(SIGFPE, ...)` 来重新激活 INT0 SIGFPE!

源 程 序

```
static void interrupt Int0Catcher(bp)
unsigned bp;
{
    register CatcherPTR    action;
    if ((action = Dispatch[2]) != SIG_IGN)
    {
        if ((action == SIG_DFL) || (action == SIG_ERR))
            exit(1);
#ifdef ANSI_CONFORMING
        Dispatch[2] = SIG_DFL;          /* Reset signal */
#endif
        (*action)(SIGFPE, FPE_INTDIV0, &bp); /* Call handler */
    }
}
```

Int4Catcher() - 溢出中断处理程序 (该过程是对包括整数算术运算例外处理 SIGFPE 的扩充的一部分。ANSI 并不限制 SIGFPE 只用于浮点数)。

--signal.c

说 明 每当用 SIGFPE 调用 `signal` 时就安装该程序。激活时该过程与 `raise(SIGFPE, ...)` 等价。对于以后程序, 该程序一直有效。如果应用程序不调用 SIGFPE 处理程序, 则使用由 C 启动代码设置的缺省动作。如果应用程序借用了 INT4, 则只能用 `signal(SIGFPE, ...)` 来重新激活 INT0 SIGFPE!

源 程 序

```
static void interrupt Int4Catcher(bp)
unsigned bp;
{
    register CatcherPTR    action;
    if ((action = Dispatch[2]) != SIG_IGN)
    {
        if ((action == SIG_DFL) || (action == SIG_ERR))
            exit(1);
#ifdef ANSI_CONFORMING
        Dispatch[2] = SIG_DFL;          /* Reset signal */
#endif
        (*action)(SIGFPE, FPE_INTOVFLOW, &bp); /* Call handler */
    }
}
```

Int5Catcher() - 用于检测 80188/80186/80286/80386 和 NEC 系列处理器上 BOUND 越界的 Int5 捕捉

程序。

-- signal.c

说明 每当用 SIGSEGV 调用 signal 时就安装该程序。对于以后程序，该程序一直有效。激活该过程时检查中断自何处产生。如果返回的 CS:IP 指向一个 BOUND 指令，则表明由该指令进行的边界检查已失败。如果返回的 CS:IP 指向 BOUND 指令的其它东西，则假设用户想调用 BIOS 打印屏幕。这时就调用旧的 BIOS 处理程序。

这种方法只有在极少数情况下才会失败。一个是在代码中执行下列指令序列（可能会有人这么用）：

```
INT 5      ; Do a prt-screen
BOUND      ; Check bounds
```

这里返回的位于中断处理程序堆栈 DOES 栈顶的 CS:IP 指向一条 BOUND 指令，但它还未执行！如果考虑“如果 BOUND 指令由 INT 执行则不会真正地越界”，将会执行得更好。但是实际上并不行，因为存在下述情况：

一个程序跳转到前导数据碰巧用与 INT5 相同的 BOUND 指令：

```
JMP DOBOUND
SOMEDATA DB CPH,5      ; Data that look like an Int 5
DOBOUND BOUND ...      ; Check bounds
```

在上例中，前面的假设就不行了。

这样，我们只能忍受这种混乱了。那么我们是把不存在的越界当作真正的，还是忽视真正的越界错误呢？那我们在这里做了些什么呢？

为了方便之故我们只能下赌注。每次都检查由返回的 CS:IP 指向的指令。如果是 BOUND 指令则调用中断处。为了避免这些情况，可以在任何 BOUND 指令前简单地加上一条 NOP。

OS/2 注意：

OS/2 的 FAPI 库用与 DOS 下采用的同样方法来处理 INT5。因此，我们不必为 OS/2&FAPI 版本进行伪代

源程序

```
static void interrupt Int5Catcher(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flags)
unsigned bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flags;
{
    register CatcherPTR action;
#define BND_OPCODE 0x62
#ifdef OS2
    /* OS/2 and FAPI correctly handle this case so we don't need to. */
    if ((*((UCHAR far * far *)((unsigned far *)&ip))) != BND_OPCODE)
        (*BiosPrtScr)(); /* Hand off Prt-Sc request */
    else
        /* Process BOUND violation */
#endif
    if ((action = Dispatch[3]) != SIG_IGN)
    {
        if ((action == SIG_DFL) || (action == SIG_ERR))
            exit(1);
        Dispatch[3] = SIG_DFL; /* Reset signal */
        (*action)(SIGSEGV, SEGV_BOUND, &bp); /* Call handler */
    }
#pragma warn -par
#pragma warn .par
}
```

Int6Catcher() - 用于检测 80188/80186/80286/80386 和 NEC 系列处理器上非法操作的 Int6 捕捉程序。

-- signal.c

说明 每当用 SIGILL 调用 signal 时就安装该程序。对于以后程序，该程序一直有效。

源程序

```
static void interrupt Int6Catcher(bp)
unsigned bp;
{
    register CatcherPTR action;
```

```

    if ((action = Dispatch[1]) != SIG_IGN)
    {
        if ((action == SIG_DFL) || (action == SIG_ERR))
        {
            exit(1);
            Dispatch[1] = SIG_DFL; /* Reset signal */
            (*action)(SIGILL, ILL_EXECUTION, &bp); /* Call handler */
        }
    }
}

```

int86 通用 8086 软中断接口

-- int86.cas

用 法 #include <dos.h>

```

    int int86(int intr_num, union REGS *inregs,
              union REGS *outregs);

```

相关函数 int int86x(int intr_num, union REGS *inregs,

用 法 union REGS *outregs, struct SREGS *segregs);

原 型 在 dos.h

说 明 这两个函数都执行一个由参数 intr_num 指定的 8086 软中断。

在执行软中断以前，两个函数都把 inregs 中的寄存器值拷贝到各寄存器中。

另外，在执行软中断前，int86x 把 segregs->x.es 和 segregs->x.ds 值拷贝到相应的寄存器中。这个特性允许程序在软中断过程中，使用远指针或大型数据存储模式，以指定要使用的段。

软中断返回后，这两个函数都把当前寄存器的值拷贝到 outregs，并把系统进位标志拷贝到 outregs 中的 x.cflag 字段中，把 8086 标志寄存器值拷贝到 outregs 的 x.flag 中。另外，int86x 恢复 DS，设置 sereg->es 和 segregs->ds 的值为对应段寄存器的值。

如果进位标志被置位，表示出现了错误。

int86x 允许激活 8086 软中断来获取一个不同于缺省数据段的 DS 值，或取一个 ES 中的参数。

注意：inregs 能指向 outregs 所指的同一结构

返 回 值 int86 和 int86x 在软中断完成后返回 AX 的值。若进位标志被置位(outregs->x.cflag!=0)，则表示出错，并把_doserrno 置为出错代码。

可移植性 只适用于 MS-DOS。int86 和 int86x 工作在 8086 系列处理器上。

参 见 intdos

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <dos.h>
#include <io.h>
int int86(int intno, union REGS *inregs, union REGS *outregs)
{
    struct SREGS s;
    segread(&s);
    return(int86x(intno, inregs, outregs, &s));
}

```

int86x 通用 8086 软中断接口

用 法 #include <dos.h>

```

    int int86x(int intr_num, union REGS *inregs,
              union REGS *outregs, struct SREGS *segregs);

```

原 型 在 dos.h

说 明 见 int86

源 程 序

```

int int86x(int intno, union REGS *inregs, union REGS *outregs,

```

```

        struct SREGS *segregs)
{
    void    (far * Vector)(void);
    char    Code[10];
    /* Save caller context */
asm    push    ds
    /* Prepare Interrupt call */
asm    lea     cx, Code
asm    mov     word ptr Vector, cx
asm    mov     word ptr Vector+2, ss
asm    mov     byte ptr Code, 055h
asm    mov     byte ptr Code+1, 0CDh
asm    mov     ax, intno
asm    mov     byte ptr Code+2, al
asm    mov     word ptr Code+3, 0CB5Dh
asm    cmp     al, 025h
asm    jb     SetRegs
asm    cmp     al, 026h
asm    ja     SetRegs
asm    mov     byte ptr Code+3, 036h
asm    mov     word ptr Code+4, 0068Fh
asm    mov     word ptr Code+6, cx
asm    mov     word ptr Code+8, 0CB5Dh
    /* Set registers with register structure content */
SetRegs:
asm    LDS     si, segregs
asm    push    [si].es
asm    push    [si].ds
asm    LDS     si, inregs
asm    mov     ax, [si].ax
asm    mov     bx, [si].bx
asm    mov     cx, [si].cx
asm    mov     dx, [si].dx
asm    mov     di, [si].di
asm    mov     si, [si].si
asm    pop     ds
asm    pop     es
    /* Call the interrupt routine */
    (* Vector)();
    /* Set register structure with registers */
asm    pushf
asm    pushf
asm    push    si
asm    push    ds
asm    push    es
asm    #if !LDATA
asm    mov     ds, [bp-20]
asm    #endif
asm    LDS     si, segregs
asm    pop     [si].es
asm    pop     [si].ds
asm    LDS     si, outregs
asm    pop     [si].si
asm    pop     [si].flags
asm    pop     [si].cflag
asm    and     word ptr [si].cflag, 1
asm    mov     [si].di, di
asm    mov     [si].dx, dx
asm    mov     [si].cx, cx
asm    mov     [si].bx, bx
asm    mov     [si].ax, ax
asm    pop     ds
asm    jz     int86Ok
asm    push    ax
asm    _IOerror(_AX);
asm    pop     ax
int86Ok:
    return(_AX);
}

```

intdos 通用 MS-DOS 中断接口

-- intdos.cas

用 法 #include <dos.h>

int intdos(union REGS *inregs, union REGS *outregs);

相关函数 int intdosx(union REGS *inregs, union REGS *outregs,

用 法 struct SREGS *segregs);

原 型 在 dos.h

说 明 这两个函数执行 DOS 中断 0x21 去激活一指定的 DOS 功能。其中 inregs->h.al 指明功能号。另外, intdosx 在调用 DOS 功能前, 把 segres->x.ds 和 segres->x.es 的值拷贝到对应的寄存器中。这一特性允许使用远指针或大型数据模式的程序在函数执行过程中指定所使用的段。

从中断 0x21 返回后, 这两个函数都拷贝当前寄存器值到 outregs 中, 拷贝系统进位标志状态到 outregs 的 x.cflag 字段中, 并拷贝 8086 标志寄存器到 outregs 的 x.flags 字段中。另外, intdosx 恢复 ds, 设置 segres->es 和 segres->ds 字段为相对段寄存器的值。

如果进位标志被置位, 表示发生了错误。

intdosx 允许激活 DOS 功能, 用来取一个不同于缺省数据段的 DS 值或取 ES 中的一个参数。

注意: inregs 可以指向由 outregs 所指的同一结构。

返 回 值 intdos 和 intdosx 在 DOS 功能调用完成后返回 AX 的值。如果进位标志被置位 (outregs->x.cflag!=0), 则表明出现了错误, 置 _doserrno 为错误代码。

可移植性 只适用于 MS-DOS

参 见 segread

源 程 序 #pragma inline

#include <asmrules.h>

#include <dos.h>

#include <io.h>

int intdos(union REGS *inregs, union REGS *outregs)

```
{
    struct SREGS s;
    segread(&s);
    return(intdosx(inregs, outregs, &s));
}
```

intdosx 通用 MS-DOS 中断接口

-- intdos.cas

用 法 #include <dos.h>

int intdosx(union REGS *inregs, union REGS *outregs,

struct SREGS *segregs);

原 型 在 dos.h

说 明 见 indos

源 程 序

#pragma warn -use

int intdosx(union REGS *inregs, union REGS *outregs, struct SREGS *segregs)

```
{
    register int SI, DI;
    /* Save caller context */
asm      push    ds
    /* Set registers with register structure content */
asm      LDS     si, segregs
asm      push    [si].es
asm      push    [si].ds
asm      LDS     si, inregs
asm      mov     ax, [si].ax
asm      mov     bx, [si].bx
asm      mov     cx, [si].cx
}
```



```

asm    mov     dx, [si].dx
asm    mov     di, [si].di
asm    mov     si, [si].si
asm    pop     ds
asm    pop     es
asm    /* Call DOS */
asm    push    bp                /* just to be safe */
asm    int     021h
asm    pop     bp
asm    /* Set register structure with registers */
asm    pushf
asm    pushf
asm    push    si
asm    push    ds
asm    push    es
asm    #if !LDATA
asm    mov     ds, [bp-6]
asm    #endif
asm    LDS     si, segregs
asm    pop     [si].es
asm    pop     [si].ds
asm    LDS     si, outregs
asm    pop     [si].si
asm    pop     [si].flags
asm    pop     [si].cflag
asm    and     word ptr [si].cflag, 1
asm    mov     [si].di, di
asm    mov     [si].dx, dx
asm    mov     [si].cx, cx
asm    mov     [si].bx, bx
asm    mov     [si].ax, ax
asm    pop     ds
asm    jz      intdosOk
asm    push    ax
asm    _IOerror(_AX);
asm    pop     ax
intdosOk:
    return(_AX);
}
#pragma warn .use

```

intr 改变软中断接口 -- intr.cas

用 法 #include <dos.h>

void intr (int intr_num, struct REGPACK *preg);

原 型 在 dos.h

说 明 intr函数用于改变执行软中断的接口，它产生一个由intr_num参数指定的8086软中断。

intr在执行软中断前，拷贝REGPACK结构preg中的寄存器值到各寄存器。在软中断完成后，intr拷贝当前寄存器值到preg中。标志被保存。

传给intr的参数如下：

intr_num 要执行的中断号

preg 结构地址，包括：

(a)调用前的输入寄存器

(b)中断调用后的寄存器值

REGPACK结构preg(在dos.h中定义)具有以下格式：

```

struct REGPACK
{
    unsigned r_ax, r_bx, r_cx, r_dx;

```

unsigned r_bp, r_si, r_di, r_ds, r_es, r_flags;

返回值 不返回值。在中断调用后，REGPACK结构preg包含了寄存器的值。

可移植性 只适用于MS-DOS，工作在8086系列处理器上

参见 int86, intdos

源程序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
void intr(int int_type, struct REGPACK *preg)
{
    void (far * Vector)(void);
    char Code[14];
    /* #define WhereIsBP      -(sizeof(Code) + sizeof(Vector) + 12)  */
#define WhereIsBP      -30
    /* Save caller context */
asm    push    bp
asm    push    ds
asm    pushf
    /* Prepare Interrupt call */
asm    lea     cx, Code
asm    mov     word ptr Vector, cx
asm    mov     word ptr Vector+2, ss
asm    mov     word ptr Code, 06E8Bh
asm    mov     byte ptr Code+2, WhereIsBP
asm    mov     byte ptr Code+3, 0CDh
asm    mov     ax, int_type
asm    mov     byte ptr Code+4, al
asm    cmp     al, 025h
asm    jnb     NormalIntr
asm    cmp     al, 026h
asm    jnb     NormalIntr
asm    mov     byte ptr Code+5, 036h
asm    mov     word ptr Code+6, 00068Fh
asm    mov     word ptr Code+8, cx
asm    mov     byte ptr Code+10, 0CAh
asm    mov     word ptr Code+11, 2
asm    jmp     SetRegs
asm    popf_proc proc near
asm    iret                     /* this proc does a bullet-proof popf */
asm    popf_proc endp
NormalIntr:
asm    mov     byte ptr Code+5, 0CAh
asm    mov     word ptr Code+6, 2
    /* Set registers with register structure content */
SetRegs:
asm    LDS     di, preg
asm    push    ds
asm    push    di
asm    mov     ax, [di].r_ax
asm    mov     bx, [di].r_bx
asm    mov     cx, [di].r_cx
asm    mov     dx, [di].r_dx
asm    push    word ptr [di].r_bp /* BP will be loaded before int xx */
asm    mov     si, [di].r_si
asm    mov     es, [di].r_es
asm    lds     di, [di].r_di
    /* Call the interrupt routine */
    (* Vector)();
    /* Set register structure with registers */
asm    push    ds
asm    push    di
asm    push    bp
asm    pushf
asm    mov     bp, sp
asm    lds     di, [bp+8] /* DS:DI points to the reg structure */
}
```

```

asm    mov    [di],r_ax,ax
asm    mov    [di],r_bx,bx
asm    mov    [di],r_cx,cx
asm    mov    [di],r_dx,dx
asm    mov    [di],r_si,si
asm    mov    [di],r_di,di
asm    mov    [di],r_bp,bp
asm    pop    [di],r_flags /* flags */
asm    pop    [di],r_bp    /* BP */
asm    pop    [di],r_di    /* DI */
asm    pop    [di],r_ds    /* DS */
asm    add    sp,4         /* Remove saved DS:DI */
asm    push   cs
asm    call   popf_proc    /* pop flags */
asm    pop    ds
asm    pop    bp
}

```

ioctl 控制I/O设备 -- ioctl.cas

用法 `int ioctl(int handle,int cmd[,int *argdx,int argc[x]]);`

原型在 `io.h`

说明 这是MS-DOS调用0X44的直接接口(IOCTL)。

确切的功能依赖于如下的cmd值:

- 0 取设备信息
- 1 设置设备信息(在 argdx)
- 2 读 argcx 个字节到 aegdx 所指的地址中
- 3 从 argdx 所指的地址中写 argcx 字节
- 4 同 2, 除了 handle 被看作是驱动器号外(0=缺省, 1=A 等)
- 5 同 3, 除了 handle 被看作驱动器外(0=缺省, 1=A 等)
- 6 取输入状态
- 7 取输出状态
- 8 测试可删除性, 只适用于 DOS 3.X
- 11 设置共享冲突的重试次数, 只适用于DOS 3.X

ioctl用于取得有关设备通道的信息。可以使用正则文件, 但只有cmd为0,

6, 7被定义, 其他所有的对文件的调用将返回EINVAL错误。

有关参数和返回值的详细信息, 请参阅《DOS程序员参考手册》中有关系统调用0X44的文档说明。

参数argdx和argcx是任选的。

ioctl为特殊函数使用DOS 2.0设备驱动程序提供一直接的接口。这个函数的功能随不同的硬件和设备而变化。而且有些机器不遵循这里描述的接口。因此, 为确切使用ioctl函数, 请参阅有关机器BIOS的文档说明。

返回值 对于cmd为0, 1, 返回值为设备信息(IOCTL调用后的DS值)。

对于为2到5, 返回值为实际传送的字节数。

对于cmd值为6, 7, 返回值为设备状态。

在任何情况下, 如果检测到一个错误, 则返回-1, 并置errno值为下列之一:

- EINVAL 无效参数
- EBADF 无效文件号
- EINVDAT 无效数据

可移植性 ioctl适用于UNIX系统, 但不是这样的参数和功能。在UNIX版本7和系统III中, ioctl的实际使用也不同。ioctl调用不能移植到UNIX中, 也很少移植到MS-DOS机器

上。

DOS 3.0扩展了ioctl, cmd值可以为8或11。

源程序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
int ioctl (fd, func, argdx, argcx)
{
    int fd; int func; void *argdx; int argcx;
    pushDS
asm    mov     ah, 44h
asm    mov     al, func
asm    mov     bx, fd
asm    mov     cx, argcx
asm    LDS     dx, argdx
asm    int     21h
asm    popDS
asm    jc      ioctlFailed
    if (func == 0)
        return(_DX); /* Dev info is in dx */
    return(_AX);
ioctlFailed:
    return __IOError (_AX);
}
```

__IOError 设置错误变量。 -- ioerror.cas

用法 #include <_io.h>
int ~ pascal __IOError (int dosErr);

原型文件 _io.h

说明 如果为负, dosErr 是 MSDOS 错误号, 即系统V错误号的相反数, __IOError用错误号设置_doserrno; 如果非负, 则将其转换为系统V的等价号。最后将转换后的值存入_errno。

返回值 -1(通常的RTL错误返回)。

源程序

```
#pragma inline
#include <asmrules.h>
#include <_io.h>
#include <errno.h>
int doserrno = 0;
char _dosErrorToSV [] =
{
    0, /* 0 - OK */
    EINVAL, /* 1 - e_badFunction */
    ENOENT, /* 2 - e_fileNotFound */
    ENOENT, /* 3 - e_pathNotFound */
    EMFILE, /* 4 - e_tooManyOpen */
    EACCES, /* 5 - e_accessDenied */
    EBADF, /* 6 - e_badHandle */
    ENOMEM, /* 7 - e_mcbDestroyed */
    ENOMEM, /* 8 - e_outOfMemory */
    ENOMEM, /* 9 - e_badBlock */
    E2BIG, /* 10 - e_badEnviron */
    ENOEXEC, /* 11 - e_badFormat */
    EACCES, /* 12 - e_badAccess */
    EINVAL, /* 13 - e_badData */
    EFAULT, /* 14 - reserved */
    EXDEV, /* 15 - e_badDrive */
    EACCES, /* 16 - e_isCurrentDir */
    ENOTSAM, /* 17 - e_notSameDevice */
    ENOENT, /* 18 - e_noMoreFiles */
    EROFS, /* 19 - e_readOnly */
    ENXIO, /* 20 - e_unknownUnit */
}
```

```

EBUSY, /* 21 e_notReady */
EIO, /* 22 e_unknownCommand */
EIO, /* 23 e_dataError */
EIO, /* 24 e_badRequestLength */
EIO, /* 25 e_seekError */
EIO, /* 26 e_unknownMedia */
ENXIO, /* 27 e_sectorNotFound */
EBUSY, /* 28 e_outOfPaper */
EIO, /* 29 e_writeFault */
EIO, /* 30 e_readFault */
EIO, /* 31 e_generalFault */
EACCES, /* 32 e_sharing */
EACCES, /* 33 e_lock */
ENXIO, /* 34 e_diskChange */
ENFILE, /* 35 e_FCBunavailable */
ENFILE, /* 36 e_sharingOverflow */
EFAULT, EFAULT,
EFAULT, EFAULT,
EFAULT, EFAULT,
EFAULT, EFAULT,
EFAULT, EFAULT,
EFAULT, EFAULT,
EFAULT, EFAULT,
ENODEV, /* 37-49 reserved */
EBUSY, /* 50 e_networkUnsupported */
EEXIST, /* 51 e_notListening */
ENOENT, /* 52 e_dupNameOnNet */
EBUSY, /* 53 e_nameNotOnNet */
ENODEV, /* 54 e_netBusy */
EAGAIN, /* 55 e_netDeviceGone */
EIO, /* 56 e_netCommandLimit */
EIO, /* 57 e_netHardError */
EIO, /* 58 e_wrongNetResponse */
EIO, /* 59 e_netError */
EINVAL, /* 60 e_remoteIncompatible */
EFBIG, /* 61 e_printQueueFull */
ENOSPC, /* 62 e_printFileSpace */
ENOENT, /* 63 e_printFileDeleted */
ENOENT, /* 64 e_netNameDeleted */
EACCES, /* 65 e_netAccessDenied */
ENODEV, /* 66 e_netDeviceWrong */
ENOENT, /* 67 e_netNameNotFound */
ENFILE, /* 68 e_netNameLimit */
EIO, /* 69 e_netBIOSlimit */
EAGAIN, /* 70 e_paused */
EINVAL, /* 71 e_netRequestRefused */
EAGAIN, /* 72 e_redirectionPaused */
EFAULT, EFAULT,
EFAULT, EFAULT,
EFAULT, EFAULT,
EFAULT,
EEXIST, /* 73-79 reserved */
EFAULT, /* 80 e_fileExists */
ENOSPC, /* 81 reserved */
EIO, /* 82 e_cannotMake */
ENFILE, /* 83 e_failInt24 */
EEXIST, /* 84 e_redirectionLimit */
EPERM, /* 85 e_dupRedirection */
EINVAL, /* 86 e_password */
EIO, /* 87 e_parameter */
/* 88 e_netDevice */

);
int pascal _IOError (int dosErr) {
asm mov si, dosErr
asm or si, si
asm jl ser_maybeSVerr
asm cmp si, e_dosFinalError
asm jna ser_dosError
ser_errorFault:
asm mov si, e_parameter
ser_dosError:

```

```

asm    mov     doserrno, si
asm    mov     al, _dosErrorToSV [si]
asm    cbw
asm    xchg    si, ax
asm    jmp     short ser_end
ser_maybeSVerr:
asm    neg     si
asm    cmp     si, _sys_nerr
asm    ja     ser_errorFault
asm    mov     _doserrno, -1          /* unknown error kind */
ser_end:
        errno = _SI;
        return -1;
}

```

is... 字符分类宏

用法 #include <ctype.h>

```

int isalpha (int ch);
int isalnum (int ch);
int isascii (int ch);
int iscntrl (int ch);
int isdigit (int ch);
int isgraph (int ch);
int islower (int ch);
int isprint (int ch);
int ispunct (int ch);
int isspace (int ch);
int isupper (int ch);
int isxdigit (int ch);

```

原型在 ctype.h

说明 这些宏通过查表对ASCII码整数值进行分类。每一个都是一谓词，true时返回非零值，false时返回零。

isascii对所有的整数值都有定义，其余的宏只是在isascii为true或ch是EOF时有定义。

返回值	isapha	非零，若ch是一字母('A'-'Z','a'-'z')
	isalnum	非零，若ch是一字母或数字('A'-'Z','a'-'z'或'0'-'9')
	isascii	非零，若ch在0-127范围内(0x00-0x7f)
	iscntrl	非零，若ch是一个删除字符或普通的控制字符(0x7f或0x00-0x1f)
	isdigit	非零，如果ch是一数字('0'-'9')
	isgraph	非零，如果ch是一打印字符，与isprint相似，除了不包括空格外(0x21-0x7E)
	islower	非零，如果ch是小写字母('a'-'z')
	isprint	非零，如果ch是打印字符(0x20-0x7E)
	ispunct	非零，如果ch是标点字符(iscntrl或isspace)
	isspace	非零，如果ch是空格、制表、回车、换行、竖向制表或格式馈送符(0x09-0x0d,0x20)
	isupper	非零，如果ch是大写字母('A'-'Z')
	isxdigit	非零，如果ch是十六进制数('0'-'9','A'-'F','a'-'f')

可移植性 所有这些宏都适用于UNIX机器。

isalpha, isdigit, islower, isspace和isupper在kernighan和Ritchie所著书中有

定义。

isatty 检查设备类型

-- isatty.cas

用 法 int isatty(int handle);

原 型 在 io.h

说 明 isatty是一确定handle代表的是下列哪一字符的函数:

- * 终端
- * 控制台
- * 打印机
- * 串行口

返 回 值 如果设备是字符设备, isatty返回一非零整数; 如果不是字符设备, 返回0。

源 程 序

```
#pragma inline
#include <io.h>
int isatty(int handle) {
asm    mov    ax, 4400h
asm    mov    bx, handle
asm    int    21h
asm    mov    ax, 0
asm    jb     DosError
asm    shi    dx, 1
asm    rcl    ax, 1
DosError:
    return _AX;
}
```

函 数 名 __isDST - 确定日数保存(daylight savings)是否有效。

-- tzset.cas

用 法 int pascal __isDST (unsigned hqur, unsigned yday,
unsigned month, unsigned year);

原型文件 _io.h

说 明 对于给定日期, 如果日数保存有效则返回非零。

如果month为0, yday 就是year 的天数; 否则yday是month的天数。

假设调用程序调用了tzset() 来填充timezone信息。

返 回 值 如果对给定日期DST有效, 则返回非0。

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <_io.h>
#include <time.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define YES 1
#define NO 0
#define Normal 0
#define Daylight 1
#define TZstrlen 3 /* Len of tz string(- null terminator) */
#define DefaultTimeZone 5L
#define DefaultDaylight YES
#define DefaultTZname "EST" /* Default normal time zone name */
#define DefaultDSTname "EDT" /* Default daylight savings zone name */ unsigned _monthDay
[] =
```

```

{ 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365
};
static char DfltZone[ TZstrlen+1 ], DfltLight[ TZstrlen+1 ];
char *const tzname[2] = {& DfltZone[0], & DfltLight[0]};
long timezone = DefaultTimeZone * 60L * 60L; /* Set for EST */
int daylight = DefaultDaylight; /* Apply daylight savings */
int pascal isDST (unsigned hour, unsigned yday, register unsigned month, unsigned year)
{
    register unsigned temp;
    if (month == 0) /* if only day of year given */
    {
        temp = yday;
        if (yday >= 31+28 && ( ((year+70) & 3) == 0)) temp--;
        for (month = 0; temp >= _monthDay[month]; month++);
    }
    else /* if month+day of month given */
    {
        if (month < 3 || ( ((year+70) & 3) != 0)) yday--;
        yday += _monthDay[month-1];
    }

asm    cmp     month, 4
asm    jb      _notDST /* has to be >= April */
asm    je      _check
asm    cmp     month, 10
asm    ja      _notDST /* has to be <= October */
asm    jne     _isDST
_check:
asm    mov     bx, month
asm    shl     bx, 1
asm    word ptr year, 16
asm    cmp     jle _else 1 /* skip if before 1986 */
asm    cmp     month, 4
asm    jne     _else 1 /* skip if not April */
asm    mov     cx, _monthDay-2[bx]
asm    add     cx, 7 /* day = 7th day in month */
asm    jmp     _endif 1
_elif 1:
asm    mov     cx, _monthDay[bx] /* day = last day in month */
_endif 1:
asm    mov     bx, year
asm    add     bx, 1970
asm    test    bl, 3 /* leap year ? */
asm    jz      _leap
asm    dec     cx /* no --> adjust */
_leap:
asm    mov     bx, year
asm    inc     bx
asm    sar     bx, 1
asm    sar     bx, 1
asm    add     bx, cx /* add leap days since 1970 */
asm    mov     ax, 365
asm    mul     word ptr year /* ax = (year-1970) * 365 */
asm    add     ax, bx
asm    add     ax, 4 /* 01-01-70 was Thursday */
asm    xor     dx, dx
asm    mov     bx, 7 /* find day of week */
asm    div     bx
asm    sub     cx, dx /* cx = threshold day of year */
asm    mov     ax, yday
asm    cmp     month, 4
asm    jne     _October
asm    cmp     ax, cx
asm    ja      _isDST /* is DST if past threshold */
asm    jne     _notDST
asm    cmp     byte ptr hour, 2
asm    jb      _notDST /* not DST if too early */
asm    jmp     _SHORT _isDST
_October:
asm    cmp     ax, cx
asm    jb      _isDST /* is DST if before threshold */
asm    jne     _notDST
asm    cmp     byte ptr hour, 1

```



```
asm    ja      _notDST      /* not DST if too late      */
_isDST:
    return(1);
_notDST:
    return(0);
}
```

itoa 把一整数转换为字符串

— itoa.cas

用 法 char *itoa (int value, char *string, int radix);

相关函数 char *ltoa(long value, char *string, int radix);

用 法 char *ultoa(unsigned long value, char *string, int radix);

原 型 在 stdlib.h

说 明 这些函数把value值转换为以空字符终结的字符串，并把结果存在string中。

对于itoa, value为整型；对于ltoa, 它为一长整型；对于ultoa, 它为一无符号长整型。

radix指明在转换value过程中的基数值，它必须在2到36之间。对于itoa和

ltoa, 如果value为负数，而radix为10, 则strssing的第一个字符为减号。在ultoa中不会有这种情况，ultoa也不进行溢出检查。

注意：分配string的空间必须足够大以容纳所返回的所有字符(包括空字符终结符\0)。itoa最多能返回17个字节，而ltoa和ultoa最多为33字节。

返 回 值 所有这些函数都返回指向string的指针，并且没有错误返回。

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <stdlib.h>
#include <printf.h>
#include <rules.h>
char *itoa (int value, char *strP, int radix) {
#define dword unsigned long
    return __longtoa ((radix == 10) ? (long) value :
                      (dword)((unsigned)value), strP, radix, true, 'a');
}
```

_KbdFlush 刷新键盘缓冲区。

-- getpass.c

用 法 static void pascal _KbdFlush(void);

说 明 刷新键盘缓冲区。

返 回 值 无。

源 程 序

```
#include <conio.h>
#include <stdio.h>
#include <dos.h>
static void pascal _KbdFlush(void)
{
    bdos(0x0c, 0x00, 0x02);
}
```

kbhit 检查当前按下的键

-- kbhit.cas

用 法 int kbhit(void);

原 型 在 conio.h

说 明 kbhit检查一按下的键当前是否有效。所有有效的按键可用getch或getche读取。

返 回 值 如果一按键有效，kbhit返回一非零整数；否则，返回0。

参 见 getch

源程序

```
#pragma inline
#include <conio.h>
#pragma warn -rvl
int kbhit(void) {
    asm    mov     ah, 0Bh
    asm    int     21h
    asm    cld
}
#pragma warn rvl
```

keep 退出并继续驻留

-- keep.c

用法 void keep(int status,int size);

原型在 dos.h

说明 keep返回MS-DOS, 把出口状态保存在status中。

当前程序仍驻留在内存中, 程序所占的存储空间为size, 内存其余部分被释放。

keep可以用于安装一个TSR程序, keep使用dos功能调用0x31。

返回值 无

可移植性 只适用于MS-DOS

源程序

```
#include <dos.h>
extern void _restorezero(void);
void keep(unsigned char status, unsigned size)
{
    _restorezero();
    _DX = size;
    _AX = status;
    _AX = 0x31;
    _getinterrupt(0x21);
}
```

labs 给出长绝对值

-- labs.c

用 法 long labs(long n);

原型在 stdlib.h

说 明 见abs

源程序

```
#include <stdlib.h>
long labs(long n)
{
    return(n < 0 ? -n : n);
}
```

ldexp 计算value*2^{exp} 的值

-- ldexp.cas

用 法 double ldexp(double value ,int exp);

原型在 math.h

说 明 见exp

源程序

```
#pragma inline
#include <asm.rules.h>
#include <math.h>
#include <math.h>
#include <errno.h>
#pragma warn -rvl
double ldexp (double value, int scale)
{
    double yVal; /* used in error exits */
    asm    FILD    W0 (scale)
```

```

/* While that is loading, we should check for range error. */
asm    mov     ax, 7FF0h
asm    and     ax, value [6]
asm    mov     cl, 4
asm    ror     ax, cl
asm    FLD     DOUBLE (value)
asm    jz      idx_zero
asm    mov     bx, scale
asm    cmp     bh, 7h
asm    jg      idx_overflow
asm    cmp     bh, -7h
asm    jl      idx_overflow
asm    add     ax, bx
asm    jng     idx_underflow
asm    cmp     ax, 7FFh
asm    jnl     idx_overflow
asm    FSCALE
idx_zero:
asm    FSTP    st(1)                /* remove the scale from the stack */
idx_end:
    return;
idx_overflow:
asm    mov     si, OVERFLOW
asm    jmp     short idx_err
idx_underflow:
asm    mov     si, UNDERFLOW
idx_err:
asm    FSTP    st(0)                /* pop value from stack */
asm    FSTP    DOUBLE (yVal)        /* yVal = scale */
#pragma warn -ret
    return matherr ( SI, "ldexp", &value, &yVal,
                    (OVERFLOW == _SI) ? HUGE_VAL : 0.0);
#pragma warn .ret
}
#pragma warn .rvl

```

ldiv 两个长整型数相除，返回商和余数 -- ldivt.cas

用 法 #include <stdlib.h>

ldiv_t ldiv(long lnumer, long ldenom);

说 明 见div

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <stdlib.h>
ldiv_t ldiv(long numer, long denom)
{
    ldiv_t ldivR;
    /* Save frame pointer */
asm    push    bp
asm    xor     cx, cx
    /* Get both operands */
asm    mov     ax, W0(numer)
asm    mov     dx, W1(numer)
asm    mov     bx, W0(denom)
asm    mov     bp, W1(denom)
    /* Zero low word of high 32 bit of 64-bit remainder */
asm    xor     si, si
    /* Signed division should be done. Convert negative values to positive
       and do an unsigned division. Store the sign value in the next
       higher bit of cl (test mask of 4). Thus when we are done, testing
       that bit will determine the sign of the result. */
    /* Negate numerator if negative number */
asm    or      dx, dx
asm    jns     PosNumer
asm    neg     dx
asm    neg     ax

```

```

asm    sbb    dx, si
asm    inc    ct
PosNumer:
/*      Negate denominator if negative number    */
asm    or     bp, bp
asm    jns    PosDenom
asm    neg    bp
asm    neg    bx
asm    sbb    bp, si
asm    xor    cl, 1
PosDenom:
/*      Save sign of the result                    */
asm    push   cx
/*      Use machine divide if high words are both zero */
asm    mov    di, bp
asm    or     di, dx
asm    jnz    noQuickDiv
asm    div    bx
asm    xchg   si, dx
asm    jmp    short commonOut
/*      Unsigned long division */
noQuickDiv:
asm    mov    cx, 32          /* Shift counter */
asm    mov    di, si          /* Fake a 64 bit dividend (clear high word) */
xLoop:
asm    shl    ax, 1           /* Shift dividend left one bit */
asm    rcl    dx, 1
asm    rcl    si, 1
asm    rcl    di, 1
asm    cmp    di, bp          /* Dividend larger? */
asm    jb     noSub
asm    ja     Subtract
asm    cmp    si, bx          /* Maybe */
asm    jb     noSub
Subtract:
asm    sub    si, bx
asm    shb    di, bp          /* Subtract the divisor */
asm    inc    ax              /* build quotient */
noSub:
asm    loop   xLoop
/*      When done with the loop the four register value look like:
      |      |      |      |
      |  di  |  si  |  dx  |  ax  |
      |-----|-----|
      | remainder | quotient | */
commonOut:
asm    pop    cx              /* Restore sign of the result */
asm    jcxz   Divided
asm    neg    di
asm    neg    si
asm    sbb    di, 0           /* Negate remainder */
asm    neg    dx
asm    neg    ax
asm    sbb    dx, 0           /* Negate quotient */
/*      Restore frame pointer, and set ldiv_t structure */
Divided:
asm    pop    bp
asm    mov    W0(ldivR), ax
asm    mov    W1(ldivR), dx
asm    mov    W0(ldivR+4), si
asm    mov    W1(ldivR+4), di
return ldivR;
}

```

ldtrunc

-- ldtrunc.cas

用 法 double pascal _ldtrunc(int flag, long double x, double xhuge);

说 明 _ldtrunc是一个内部的TC库函数，它在库函数所能工作的限制范围内将一个 long double 型数截断为float(flag=0)或double(flag=1)型的数。参数用于下溢检查，如下溢它将被刷

新为0；如果上溢则改变为xhuge。这两种情况下，x的符号都与结果相同；否则x返回时将不改变。如果上溢或下溢，全局的errno都将被设置为ERANGE。不会产生额外的上溢或下溢。

如flag = 0, 表示xhuge = 1/0;

如flag = 1, 表示xhuge = HUGE_VAL。

源程序

```
#pragma inline
#pragma warn -rvl
#pragma warn -ret
#include <errno.h>
double pascal _ldtrunc(int flag, long double x, double xhuge)
{
    unsigned cword, cword2;
    CX = 0;
    /* AX = overflow threshold, BX = underflow threshold */
    asm    mov    ax, 43FEh
    asm    mov    bx, 3F6Ah
    asm    cmp    word ptr flag, 0
    asm    jne    start
    asm    mov    ax, 407Eh
    asm    mov    bx, 3F6Ah
start:
    asm    mov    dx, x[8]
    asm    shl    dx, 1
    asm    rcl    cx, 1
    asm    shr    dx, 1
    /* CX = sign bit, DX = biased exponent */
    /* let INF, NAN pass */
    asm    cmp    dx, 7FFFh
    asm    je     ret
    /* test for overflow */
    asm    cmp    dx, ax
    asm    je     hugex
    asm    jle    notinf
    /* overflow to HUGE_VAL with appropriate sign */
    asm    fld    qword ptr xhuge
    goto ret1;
hugex:
    /* chop in borderline infinite case, to avoid overflow */
    asm    fstcw  cword
    asm    mov    ax, 0C00h
    asm    fwait
    asm    or     ax, cword
    asm    mov    cword2, ax
    asm    fldcw  cword2
    asm    fld    tbyte ptr x
    asm    cmp    word ptr flag, 0
    asm    jne    s1
    asm    fstp   dword ptr xhuge
    asm    fld    dword ptr xhuge
    asm    jmp    short s2
s1:
    asm    fstp   qword ptr xhuge
    asm    fld    qword ptr xhuge
s2:
    /* restore previous control word */
    asm    fldcw  cword
    return;
notinf:
    /* test for +0 or -0 */
    asm    mov    ax, dx
    asm    or     ax, x[6]
    asm    or     ax, x[4]
    asm    or     ax, x[2]
    asm    or     ax, x[0]
    asm    jz     ret
```

```

/* test for underflow */
asm    cmp     dx, bx
asm    jg     ret
/* underflow to 0 with appropriate sign */
asm    fldz
retl;
asm    or     cx, cx
asm    jz     ret0
asm    fchs
ret0:
        errno = ERANGE;
        return;
ret:
        return x;
}

```

bfind 执行一线性搜索

-- **bsearch.c**

用 法 void *bfind (void *key ,void *base ,int *nelem,
int width, int(*fcmp)());

原 型 在 **stdlib.h**

说 明 见**bsearch**

源 程 序

```

#include <stdlib.h>
#include <mem.h>
#include <stddef.h>
void *bfind(const void *key, const void *base, size_t *nelem, size_t width,
            int cdecl (*fcmp)(const void *, const void *))
{
    return(_bsearch(key,base,nelem,width,fcmp,0));
}

```

line 在指定两点间画一直线

用 法 #include <graphics.h>
void far line (int x0, y0, int x1, int y1);

相关函数 void far lineto (int dx ,int dy);

用 法 void far linerel(int dx, int dy);

原 型 在 **graphics.h**

说 明 这些函数用当前颜色、当前线型和宽度画一直线。

line在指定两点(x0,y0)和(x1,y1)之间画一直线，当前位置(cp)不变。

lineto从cp到(x,y)画一直线，并将cp移到(x,y)。

linerel从cp到与cp相对距离为(dx,dy)的点画一直线，同时，cp增量(dx,dy)。

返 回 值 无

可移植性 在Turbo pascal 5.0中有相似的程序

参 见 **getcolor, getlinesettings**

linerel 从当前位置(cp)到与cp有一相对距离的点画一直线

用 法 #include <graphics.h>
void far linerel (int dx,int dy);

原 型 在 **graphics.h**

说 明 见**line**

lineto 从cp到(x,y)画一直线

用 法 `#include <graphics.h>`
`void far lineo(int x, int y);`
原 型 在 `graphics.h`
说 明 见 `line`

localtime 把日期和时间转变成结构 -- `ctime.c`

用 法 `#include <time.h>`
`struct tm *localtime(long *clock);`

原 型 在 `time.h`

说 明 见 `ctime`

源 程 序

```
#include <io.h>
#include <time.h>
#include <stdio.h>
static const char Days[12] = {
    31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};
static const char *const Weekday[7] = {
    "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
};
static const char *const Months[12] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
static struct tm tm;
struct tm *localtime(const long *clock)
{
    long x;
    tzset(); /* get timezone info */
    x = *clock - timezone;
    return(comtime(x, 1));
}
```

lock 设置文件共享锁 -- `lock.cas`

用 法 `int lock(int handle, long offset, long length);`

相关函数

用 法 `int unlock(int handle, long offset, long length);`

原 型 在 `io.h`

说 明 `lock`和`unlock`为MS-DOS 3.X的文件共享机制提供一个接口

`lock`能放于文件的任一非重叠区域。一个试图向锁住区进行读/写的程序将重试三次操作，如果三次都失败了，则调用失败，出现错误。

`unlock`解除`lock`；为避免出错，在文件关闭之前必须解除。程序在执行完之前，必须释放所有的锁。

返 回 值 在成功时，这两个函数都返回0，出错时，返回-1。

可移植性 只适用于MS-DOS 3.x。早期版本的MS-DOS不支持这些调用。

参 见 `open`函数

源 程 序

```
#pragma inline
#include <io.h>
#include <io.h>
int lock(int handle, long offset, long length) {
    asm mov ax, 05C00h
    asm mov bx, handle
    asm mov cx, offset + 2
    asm mov dx, offset
```

```

asm    mov     si, length+2
asm    mov     di, length
asm    int     021h
asm    jc      lockFailed
        return(0);
lockFailed:
        return _IOerror (_AX);
}

```

log 对数函数ln(x)

-- log.cas

用 法 double log(double x);

原 型 在 math.h

说 明 见exp

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#include <errno.h>
#include <stddef.h>
static unsigned short NANLOG [4] = {0,0,0x0480, 0xFFFF0};
#pragma warn -rvl
#pragma warn -use
double log (double x) {
    register SI;
    double temp;
asm    FLD     DOUBLE (x)
asm    mov     ax, W0 (x [6])          /* get the exponent field */
asm    shl     ax, 1
asm    jz      log_zero
asm    jc      log_imaginary
asm    cmp     ax, 0FFE0h
asm    je      log_infinite
asm    _FAST_  (_FLOG_)
log_end:
        return;
log_zero:
asm    mov     si, SING
        temp = -HUGE_VAL;
asm    jmp     short log_complain
log_infinite:
asm    mov     si, OVERFLOW
        temp = HUGE_VAL;
asm    jmp     short log_complain
log_imaginary:
asm    mov     si, DOMAIN
        temp = *((double *) NANLOG);
log_complain:
asm    FSTP    ST(0)                  /* pop x from stack */
#pragma warn -ret
        return _matherr (_SI, "log", &x, NULL, temp);
#pragma warn .ret
}
#pragma warn .rvl
#pragma warn .use

```

log10 对数函数log10(x)

-- log10.cas

用 法 double log10(double x);

原 型 在 math.h

说 明 见exp

源 程 序


```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <math.h>
#include <errno.h>
#include <stddef.h>
static unsigned short NANLOG [4] = {0,0,0x0480, 0xFFFF0};
#pragma warn -rvl
#pragma warn -use
double log10 (double x) {
register SI;
double temp;
asm FLD DOUBLE (x)
asm mov ax, W0 (x [6]) /* get the exponent field */
asm shl ax, 1
asm jz l10_zero
asm jc l10_imaginary
asm cmp ax, 0FFE0h
asm je l10_infinite
asm _FAST_ ( FLOG10_)
l10_end:
return;
l10_zero:
asm mov si, SING
temp = -HUGE_VAL;
asm jmp short l10_complain
l10_infinite:
asm mov si, OVERFLOW
temp = HUGE_VAL;
asm jmp short l10_complain
l10_imaginary:
asm mov si, DOMAIN
temp = *((double *) NANLOG);
l10_complain:
asm FSTP ST(0) /* pop x from stack */
#pragma warn -ret
return _matherr ( _SI, "log10", &x, NULL, temp);
#pragma warn .ret
}
#pragma warn .rvl
#pragma warn .use

```

longjmp 执行非局部转移 -- setjmp.cas

用 法 #include <setjmp.h>
void longjmp(jmp_buf env,int val);

相关函数

用 法 int setjmp(jmp_buf env);

原 型 在 setjmp.h

说 明 setjmp捕获所有的任务状态到env中，返回0。而后用此env来调用longjmp，恢复所捕获的任务状态，返回setjmp的值val。

在调用longjmp前必须先调用setjmp。在longjmp调用前，调用setjmp和设置env的例行程序必须是活动的，而且不能返回值，否则，结果将是不可预测的。

任务状态为：

- * 所有的段寄存器值(cs,ds,es,ss)
- * 寄存器变量(si,di)
- * 堆栈指针(sp)
- * 框架指针(fp)
- * 标志

一个任务状态对于setjmp和longjmp用于实现合作例程是足够的。

这些例程对于处理程序的低级子程序中遇到的错误和一些例外情况是非常有用的。

返回值 在初次调用时, setjmp返回0。

longjmp不返回0; 如果传给val, longjmp将返回1。

可移植性 适用于UNIX系统

参 见 ctrl brk, signal, raise

源 程 序

```
#pragma inline
#include <asmruses.h>
#include <setjmp.h>
void longjmp(jmp_buf jmpb, int retval)
{
    /* sneaky way to do if (retval==0) retval=1; */
    asm mov     bx,retval
    asm cmp     bx,1          /* generates carry if bx=0 */
    asm adc     bx,0          /* if was 0, now it's 1 */
    asm LDS     si, jmpb
    asm cld
    /* Change context begins with changing stack */
    asm lodsw
    asm mov     ss, [si]      /* sp */
    asm mov     sp, ax        /* SS */
    asm lodsw          /* skip SS */
    /* Build the return-link to the caller of setjmp */
    asm lodsw          /* FL */
    asm push    ax
    asm lodsw          /* CS */
    asm push    ax
    asm lodsw          /* IP */
    asm push    ax
    /* Restore other working registers */
    asm lodsw          /* BP */
    asm mov     bp, ax
    asm lodsw          /* DI */
    asm mov     di, ax
    asm lodsw          /* ES */
    asm mov     es, ax
    asm lodsw          /* SI */
    asm mov     ds, [si]
    asm mov     si, ax
    asm mov     ax, bx        /* put result in ax */
    asm iret               /* return to original caller of setjmp */
}
```

lowvideo 选择低亮度字符 -- color.c

用 法 void lowvideo(void);

原 型 在 conio.h

说 明 见highvideo

源 程 序

```
#include <_video.h>
#include <conio.h>
#define INTENSE 0x08
void lowvideo(void)
{
    _video.attribute &= ~INTENSE;
}
```

_lrotl 将无符号长整型数向左循环移位 -- lrotl.cas

用 法 unsigned long _lrotl(unsigned long lvalue, int count);

原 型 在 stdlib.h

说 明 见_rotl

源程序

```
#pragma inline
#include <asmrules.h>
#include <stdlib.h>
#pragma warn -rvl
unsigned long _lrotl(unsigned long val, int rotate_count) {
asm    mov    ax, W0(val)
asm    mov    dx, W1(val)
asm    mov    cx, rotate_count
asm    and    cx, 01Fh
asm    jz     Rotated
Rotating:
asm    rcl    ax, 1
asm    rcl    dx, 1
asm    adc    ax, 0
asm    loop   Rotating
Rotated:
    return;
}
#pragma warn .rvl
```

_lrotr 将无符号长整型数向右循环移位 -- lrotr.cas

用法 `undigned long _lrotr(unsigned long lvalue, int count);`

原型在 `stdlib.h`

说明 见 `_rotl`

源程序

```
#pragma inline
#include <asmrules.h>
#include <stdlib.h>
#pragma warn -rvl
unsigned long _lrotr(unsigned long val, int rotate_count) {
asm    mov    ax, W0(val)
asm    mov    dx, W1(val)
asm    mov    cx, rotate_count
asm    and    cx, 01Fh
asm    jz     Rotated
Rotating:
asm    mov    bx, dx
asm    shr    bx, 1
asm    rcr    ax, 1
asm    rcr    dx, 1
asm    loop   Rotating
Rotated:
    return;
}
#pragma warn .rvl
```

_lsearch 搜索一个表。 -- lsearch.c

用法 `static void *nea,`

```
    pascal _lsearch(const void      *key,
                      void          *base,
                      size_t        *nelem,
                      size_t        width,
                      int cdecl     (*fcmp)(const void *, const void *),
                      int           flag)
```

说明 根据 `flag` 的值决定执行 `lfnd` 还是 `lsearch`。如 `flag` 为 1, 在不匹配时修改表; 如 `flag` 为 0, 则只进行搜索

返回值 无。

源程序

```
#include <stdlib.h>
#include <mem.h>
#include <stddef.h>
static void *near pascal _lsearch(const void *key,
                                   register void *base,
                                   size_t *nelem,
                                   size_t width,
                                   int cdecl (*fcmp)(const void *, const void *),
                                   int flag)
{
    register int Wrk;
    for (Wrk = *nelem; Wrk > 0; Wrk--)
    {
        if (((*fcmp)(key, base)) == 0)
            return(base);
        ((char *)base) += width;
    }
    if (flag)
    {
        (*nelem)++;
        movmem(key, base, width);
    }
    else
        base = NULL;
    return(base);
}
```

lsearch 线性搜索

-- lsearch.c

用法 #include <stdlib.h>

```
void *lsearch(const void *key, void *base, size_t *nelem,
              size_t width, int (*fcmp)(const void *, const void *));
```

原型在 stdlib.h

说明 见bsearch

源程序

```
#include <stdlib.h>
#include <mem.h>
#include <stddef.h>
void *lsearch(const void *key, void *base, size_t *nelem, size_t width,
              int cdecl (*fcmp)(const void *, const void *))
{
    return(_lsearch(key, base, nelem, width, fcmp, 1));
}
```

lseek 移动文件读/写指针

-- lseek.cas

用法 #include <io.h>

```
long lseek(int handle, long offset, int fromwhere);
```

相关函数

用法 longtell(int handle)

原型在 io.h

说明 lseek把与handle相关联的文件指针移到由fromwhere所指的地址加上offset字节的新位置。fromwhere值必须为0,1,2之一, 分别代表三个符号常量, 它们在stdio.h中定义如下:

fromwhere	文件位置
SEEK_SET	(0) 文件开始
SEEK_CUR	(1) 当前文件指针位置
SEEK_END	(2) 文件结尾

tell取与handle相关联的文件指针的当前位置, 它是从文件开始处到该位置的

字节数。

返回值 lseek返回指针新位置的偏移量，它是从文件开始处算起的。在出错时，lseek返回-1，且置error为下列值之一：

EBADF 无效文件号

EINVAL 无效参数

如果设备没有查找的能力（如终端和打印机），则返回值没有定义。

tell返回当前文件指针的位置。返回-1表示发生了错误，此时，error被置为：

EBADF 无效文件号

可移植性 这些函数适用于UNIX系统

参 见 fopen, fseek, ftell, getc, setbuf, ungetc

源 程 序

```
#pragma inline
#include <io.h>
#include <fcntl.h>
#include <io.h>
long lseek (int fd, long offset, int kind)
{
    _openfd [fd] &= ~_O_EOF;          /* forget about ^Z */
    asm    mov    ah, 42h
    asm    mov    al, kind
    asm    mov    bx, fd
    asm    mov    cx, offset+2
    asm    mov    dx, offset
    asm    int    21h
    asm    jc     lseekFailed
#pragma warn -rvi
    return;
#pragma warn .rvi
lseekFailed:
#pragma warn -ret
    return  __IOerror (_AX);
#pragma warn .ret
}
```

ltoa 把一个长整型转换为字符串 -- ltoa.cas

用 法 char *ltoa(long value, char *string, int radix);

原 型 在 stdlib.h

说 明 见 itoa

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <stdlib.h>
#include <_printf.h>
#include <rules.h>
char *ltoa (long value, char *strP, int radix)
{
    return __longtoa (value, strP, radix, (radix == 10), 'a');
}
```

malloc 分配主存 -- malloc.c

用 法 void *malloc(unsigned size);

相关函数 void calloc(unsigned nelem, unsigned elsize);

用 法 unsigned coreleft (void); 对于微型、小型和中型模式

unsigned long coreleft (void); 对于紧缩、大型和巨型模式

void free(void *ptr);

void *realloc (void *ptr, unsigned newsize);

原型在 stdlib,alloc.h

说明 这些函数提供C存储堆存取手段。

堆用于建立长度可变的存储块的动态存储分配。许多数据结构，如树和表都自然地使用堆存储分配。

除了顶部紧接着的256字节边界外，数据段末尾与程序之间的所有空间都在小数据模式中使用。边界用于应用程序扩充堆栈和MS-DOS所需的一小量存储。

在大型数据模式中，程序堆栈以上直到物理存储器顶端间的所有空间都被堆所使用。

返回一指向长度为size的存储块指针。如果没有足够的存储区，malloc返回NULL，块内容不变。

calloc象malloc一样分配块，只是块的长度nelem乘以elsize。块被清零。

coreleft返回未使用的存储量，根据所使用的存储模式是小数据模式还是大数据模式，它所返回的值不一样。

free释放先前所分配的块。ptr必须包含所分配块的首地址。

realloc调整原先分配块的大小为newsize，如果需要的话，把内容拷贝到新地址中。

返回值 malloc和calloc返回新分配块的指针，如果没有足够的空间分配给新块，就返回NULL。

realloc返回重分配块的地址，它可能跟原来的地址不同。如果块不能被重新分块，realloc返回NULL。

在大数据模式中，coreleft返回堆和栈之间未用空间大小。

在小数据模式中，coreleft返回堆和栈之间未用空间大小减去256字节。

可移植性 calloc, free, malloc和realloc适用于UNIX系统。calloc在kernighan和Ritchie书中定义。

参见 allocmem, farmalloc, setbuf

源程序

```
#if defined( __TINY__ ) || defined( __SMALL__ ) || defined( __MEDIUM__ )
#include <stdio.h>
#include "heap.h"
struct header * first, * last, * rover;
/* removes a block from the free-block queue */
void cdecl pull_free_block( struct header *q )
{
    struct header *p;
    if( __rover == q->next_free ) == q )
    {
        __rover = NULL;
    }
    else
    {
        p = q->prev_free;
        rover->prev_free = p;
        p->next_free = __rover;
    }
}

/* breaks up a large block into two pieces,
one for allocating and one which remains free
returns a pointer to the allocated block */
static void *cdecl allocate_partial_block( struct header *q, unsigned int len )
{
    struct header *n;
    q->size -= len;
    n = (struct header *)((char *)q + q->size);
    n->size = len + 1;
    n->prev_real = q;
    if( __last == q )
    {
        __last = n;
    }
    else
    {
        q = (struct header *)((char *)n + len);
        q->prev_real = n;
    }
    return( (void *)&n->prev_free );
}
```

```

/* attempts to extend the heap by pushing the brk level up
returns a pointer to the last block if successful,
NULL if not
*/
static void *cdecl extend_heap( unsigned int len )
{ struct header *q;
  q = sbrk( (long)len & 0xffff );
  if( (int)q == -1 ) return( (void *)NULL );
  q->prev_free = last;
  q->size = len + 1;
  last = q;
  return( (void *)(&last->prev_free) );
}

/* creates a heap from scratch
returns a pointer to the first block if successful
NULL if not
*/
static void *cdecl create_heap( unsigned int len )
{ struct header *q;
  q = sbrk( (long)len & 0xffff );
  if( (int)q == -1 ) return( (void *)NULL );
  first = q;
  last = q;
  q->size = len + 1;
  return( (void *)(&q->prev_free) );
}

/* attempts to allocate a given number of contiguous bytes on the heap
returns a pointer to the first byte of user space within the allocated block
or NULL if not enough space was available
*/
void *cdecl malloc( unsigned int nbytes )
{ struct header *q;
  if( nbytes == 0 ) return( NULL );
  /* add the header size and force an 8-byte boundary */
  nbytes = (nbytes + USED_HEADER_SIZE + 7) & 0xffff;
  /* create a memory chain if it doesn't yet exist */
  if( ! first )
  { return( create_heap( nbytes ) );
  }

  /* search for a free block through the memory chain */
  if( (q = _rover) != 0 )
  { do
    { /* big enough to break up? */
      if( q->size >= nbytes + FREE_HEADER_SIZE + DELTA_FACTOR )
      { return( allocate_partial_block( q, nbytes ) );
      }
      /* big enough to allocate? */
      if( q->size >= nbytes )
      { _pull_free_block( q );
        ++q->size; /* mark it as used */
        return( (void *)(&q->prev_free) );
      }
      q = q->next_free;
    } while( q != _rover );

    /* couldn't find a free block big enough, try to extend the heap */
    return( extend_heap( nbytes ) );
  }
}
#endif

```

lsetmem 将一值赋予内存。

-- fcalloc.cas

用法 void near pascal lsetmem(char far *p, unsigned n,
unsigned char val);

说明 将由远指针P指向的内存块的前N个字节设置为字符val。

返回值 无。

源程序

```

#pragma inline
#include <asmrules.h>
#include <alloc.h>
#include <stddef.h>
#if (LDATA)
#include <mem.h>
#else
static void near pascal lsetmem(char far *p, unsigned n, unsigned char val) {
asm    les    di, dword ptr p
asm    mov    cx, n
asm    mov    al, val
asm    rep    stosb
}
#endif

```

_matherr 浮点运算处理程序

_matherr.c

用 法 #include <math.h>

```

double _matherr (_mexcep why, char *fun,
double *arg1p, double *arg2p, double retval);

```

相关函数 #include <math.h>

用 法 int matherr(struct exception *e);

原型在 math.h

说 明 _matherr提供对所有数学库函数中的错误进行处理的过程；它调用matherr并处理matherr的返回值。_matherr不能被用户程序直接调用。

每当数学库中例程出现错误时，_matherr就被调用，它有几个参数。

_matherr做以下四件事：

- 把其使用参数填入exception结构中。
- 使用指向exception结构的指针e来调用matherr，看matherr能否处理错误。
- 检查matherr的返回值：

如果matherr返回0(表示matherr不能处理错误)，matherr置errno值(见下面所述)并打印一出错信息。

如果matherr返回非0(表示matherr能处理错误)，_matherr什么也不干，不置errno值，也不打印任何信息。

它返回e->retval到初始调用程序。注意：matherr可以修改e->retval的值，以指定它要传给初始调用的值

当matherr设置errno时(如果matherr返回0)，它把错误类型(exception结构中的type字段)映射到errno的EDOM或ERANGE值。

返 回 值 _matherr返回值e->retval。该值开始为输入参数中任给_matherr的retval,可以被matherr修改。

如果数学函数结果大于MAXDOUBLE，retval缺省为带合适符号的HUGE_VAL宏，然后传给_matherr。如果数学函数的结果小于MINDOUBLE，retval被置为0。然后传给_matherr。在这两种情况下，matherr不修改e->retval，_matherr设置errno为：

ERANGE 结果超出范围

参 见 matherr

源 程 序

```

#include <math.h>
#include <stdio.h>
#include <errno.h>
#ifdef UNIX matherr
char *whyS [] =

```



```

        "argument domain error",
        "argument singularity ",
        "overflow range error ",
        "underflow range error",
        "total loss of significance",
        "partial loss of significance"
    );
}
double _matherr( _mexcep why,
                 char *fun,
                 double *arg1p,
                 double *arg2p,
                 double retval)
{
    struct exception e;
    e.type = why;
    e.name = fun;
    e.arg1 = (NULL == arg1p) ? 0 : *arg1p;
    e.arg2 = (NULL == arg2p) ? 0 : *arg2p;
    e.retval = retval;
    if (matherr (& e) == 0)
    {
        fprintf (stderr, "%s (%8g,%8g): %s\n", fun,
                 *arg1p,
                 *arg2p,
                 whyS [why - 1]);
        errno = ((OVERFLOW == why) || (UNDERFLOW == why)) ? ERANGE : EDOM;
    }
    return e.retval;
}
#else
char *whyS [] =
{
    "DOMAIN",          /* argument domain error -- log (-1) */
    "SING",             /* argument singularity -- pow (0,-2) */
    "OVERFLOW",        /* overflow range error -- exp (1000) */
    "UNDERFLOW",       /* underflow range error -- exp (-1000) */
    "TLOSS",           /* total loss of significance -- sin(10e70) */
    "PLOSS"            /* partial loss of signif. -- not used */
};
double _matherr( _mexcep why,
                 char *fun,
                 double *arg1p,
                 double *arg2p,
                 double retval)
{
    struct exception e;
    e.type = why;
    e.name = fun;
    e.arg1 = (NULL == arg1p) ? 0 : *arg1p;
    e.arg2 = (NULL == arg2p) ? 0 : *arg2p;
    e.retval = retval;
    if (matherr (& e) == 0)
    {
        fprintf (stderr, "%s: %s error\n", fun, whyS [why - 1]);
        errno = ((OVERFLOW == why) || (UNDERFLOW == why)) ? ERANGE : EDOM;
    }
    return e.retval;
}
#endif

```

matherr 用户可修改的数学错误处理程序 -- matherr.c

用 法 #include <math.h>

int matherr(struct exception *e);

原 型 在 math.h

说 明 Turbo缺省版本的matherr函数只是简单地返回0;它提供一种方法,用户可以用自己编写的数学错误处理程序来代替它。请看以下用户定义的matherr实现例子:

用户可以修改matherr成为自己的错误处理例程(如捕获且处理某一类型的错误);修改后的

matherr在不能处理错误时, 应该返回0, 否则返回非0值。当matherr返回非0值时, 不打印错误信息, 也不改变errno。

以下是在math.h中定义的exception结构:

```
struct exception {
    int type;
    char name;
    double arg1, arg2, retval;
};
```

exception结构的成员在下表中说明:

成员	意 义
type	数字错误类型, enum 类型由 typedef _mexcep 定义(见下一表)
name	指向引起错误的数学库函数名(以空字符为终结符)的指针
arg1	引起错误的数学函数(由 name 所指)的参数, 如果只有一个参数, 存在 arg1 中
retval	matherr 的缺省返回值, 用户可以修改

typedef

_mexcep(也在math.h中定义)

列举了表示各种可能错误的符号常量如下表:

符号常量	数字错误
DOMAIN	参数不在函数定义域内(如 log(-1))
SING	参数引起异常情况(如 pow(0,-2))
OVERFLOW	参数使函数结果大于 MAXDOUBLE(如 exp(1000))
UNDERFLOW	参数使函数结果小于 MINDOUBLE(如 exp(-1000))
TLOSS	参数使函数结果失去最高位(如 SIN(10 e 70))

符号常量MAXDOUBLE和MINDOUBLE在vaule.h中定义。

注意: matherr函数不能被修改。matherr函数广泛应用于各种C运行库中, 因此应该具有较好的可移植性。

UNIX风格matherr的缺省动作(打印信息和终止)与ANSI标准不兼容。如果需要UNIX风格的matherr版本, 可使用Turbo C源盘上提供的matherr.c文件。

返回值 matherr的缺省返回值为0, 它也能修改e->retval, 后通过_matherr传递给初始调用者。

当matherr返回0时(表示它不能处理错误), _matherr设置errno并打印错误信息(参阅_matherr)。

当matherr返回非0时(表示它能处理错误), _matherr不置errno, 也不打印信息。

源 程 序

```
#include <math.h>
#ifdef UNIX matherr
#include <stdio.h>
#include <process.h>
char *whyS [] =
{
    "argument domain error",
    "argument singularity ",
    "overflow range error ",
    "underflow range error",
    "total loss of significance",
    "partial loss of significance"
```

```

};
int matherr (struct exception *e)
{
    fprintf (stderr,
        "%s (%8g,%8g): %s\n", e->name, e->arg1, e->arg2, whyS [e->type - 1]);
    exit (1);
}
#else
int matherr(struct exception *e)
{
    if (e->type == UNDERFLOW)
    {
        /* flush underflow to 0 */
        e->retval = 0;
        return 1;
    }
    if (e->type == TLOSS)
    {
        /* total loss of precision, but ignore the problem */
        return 1;
    }
    /* all other errors are fatal */
    return 0;
}
#endif

```

max 取两个数中较大者

用 法 `int max(int a, int b);`

相关函数

用 法 `int min(int a, int b);`

原 型 在 `stdlib.h`

说 明 `max`为一宏，用于得到两个整数中较大的一个。

`min`为一宏，用于得到两个整数中较小的一个。

mem... 对存储区数组进行操作

-- `memcpy.cas`
 -- `memchr.cas`
 -- `memcmp.cas`
 -- `memcpy.cas`
 -- `memset.cas`
 -- `memcmp.c`
 -- `movmem.cas`

相关函数 `void *memcpy(void *destin, void *source,
 unsigned char ch, unsigned n);`

用 法 `void * memchr(void *s, char ch, unsigned n);`
`void * memcmp(void *s1, void *s2, unsigned n);`
`void *memcpy (void *destin, void *source, unsigned n);`
`int memicmp (void *s1, void *s2, unsigned n);`
`void *memmove(void *destin, void *source, unsigned n);`
`void *memset (void *s, char ch, unsigned n);`

原 型 在 `string.h, mem.h`

说 明 所有这些**mem...**系列的函数都对存储区数组进行操作，数组的长度为n字节。
`memcpy`从源source中拷贝n个字节到目标destin中。如果源与目标重叠，将
 选择拷贝方向，使重叠处字节能正确拷贝。

`memmove` 同 `memcpy`

`memset` `memset` 设置s中的所有字节为ch,s数组的大小由n给定

memcpy 比较两个长度均为 n 的字符串 s1 和 s2，把字节看成是无符号字符型，因此，memcpy("\xFF","\x7F",1)将返回一大于 0 的值

memcmp 比较两个串 s1 和 s2 的前 n 个字节，忽略大小写。

memccpy 从源 source 拷贝字节到目标 destn 中，直到下列情况之一发生，拷贝停止：

- 字符 ch 被首先拷贝到 destn 中
- 已经拷贝了 n 个字节

memchr 在数组的前 n 个字节中搜索字符 ch

返回值 memmove 和 memcpy 返回 destn

memset 返回 s 的值

memcmp 和 memcmp 返回

<0 如果 s1 < s2

=0 如果 s1 = s2

>0 如果 s1 > s2

如果 ch 已被拷贝，memccpy 返回指向 destn 中紧跟 ch 以后的字符的指针；否则返回 NULL。

memchr 返回指向 s 中首次出现 ch 的指针，如果 ch 没在 s 数组中出现，返回 NULL。

可移植性 适用于 UNIX 系统 V

参 见 str...

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <mem.h>
#pragma warn -rvl
void *memccpy(void *dst, const void *src, int val, size_t n) {
    #if !(LDATA)
        ES = DS;
    #endif
    asm LES di, src /* first check for any occurrence of val */
    asm mov cx, n
    asm jcxz ccq_NULL
    asm mov bx, cx
    asm mov al, val
    asm cld
    asm repne scash
    pushDS
    asm sub bx, cx /* span includes ch if it was seen */
    asm mov cx, bx /* CX = span to be copied */
    asm LES di, dst
    asm LDS si, src
    asm shr cx, 1
    asm rep movsw
    asm jnc ccq_end
    asm movsh
ccq_end:
    popDS
    #if (LDATA)
        asm mov dx, es
    #endif
    asm cmp al, ES [di-1] /* was a ch match found? */
    asm mov ax, di
    asm je ccq_exit
ccq_NULL:
    asm xor ax, ax
    #if (LDATA)
        asm mov dx, ax
    #endif
}
```

```

#endif
ccp_exit; }
#pragma warn rvl
void *memchr(const void *s, int val, size_t n) {
#ifdef !LDATA
    _ES = _DS;
#endif
asm    LES     di, s
asm    mov     cx, n
asm    jcxz    mch_NULL
asm    mov     al, val
asm    cld
asm    repne   scasb
asm    je      mch_OK
mch_NULL:
#ifdef !LDATA
asm    xor     di, di
asm    mov     es, di
#endif
asm    mov     di, 1
mch_OK:
asm    dec     di
#ifdef LDATA
    return (void *) _DI;
#else
    return (void *) _DI;
#endif
}
int memcmp(const void *s1, const void *s2, size_t n)
{
    pushDS
asm    mov     cx, n
asm    jcxz    mem_trivial
asm    LDS     si, s1
asm    LES     di, s2
#ifdef !LDATA
asm    mov     ax, ds
asm    mov     es, ax
#endif
asm    cld
asm    rep     cmpsb
/* The result is the UNSIGNED difference of the final character pair,
   be they equal or different. */
asm    mov     al, [si-1]
asm    xor     al, ah
asm    mov     cl, ES_ [di-1]
asm    xor     ch, ch
asm    sub     ax, cx mem_end:
    popDS
    return;
mem_trivial:
asm    xor     ax, ax
asm    jmp     short mem_end
}
#pragma warn rvl
#ifdef 1 /* No overlap checking version */
#pragma inline
#include <asmrules.h>
void *memcpy(void *dst, const void *src, size_t n) {
#ifdef !LDATA
    _ES = _DS;
#endif
#ifdef defined(__LARGE__) || defined(__COMPACT__)
asm    mov     dx, ds /* save ds */
#endif
asm    LES     di, dst
asm    LDS     si, src
asm    mov     cx, n
asm    shr     cx, 1
asm    cld

```

```

asm      rep      movsw
asm      jnc      cpy_end
asm      movsb
cpy_end:
#ifdef   defined(_LARGE_) || defined(_COMPACT_)
asm      mov      ds, dx      /* restore */
#endif
return(dst);
}
#else      /* Overlap checking version */
void *memcpy(void *dst, const void *src, size_t n)
{
    movmem(src, dst, n);
    return(dst);
}
#endif
int memicmp(const void *s1, const void *s2, size_t n)
{
    int dif;
    for (; n-- > 0; ((unsigned char *)s1)++, ((unsigned char *)s2)++)
    {
        dif = toupper(*(unsigned char *)s1) - toupper(*(unsigned char *)s2);
        if (dif != 0)
            return(dif);
    }
    return(0);
}
void *memmove(void *dst, const void *src, size_t n)
{
    movmem(src, dst, n);
    return(dst);
}
void *memset(void *src, int c, size_t n)
{
    setmem(src, n, c);
    return(src);
}

```

mkdir 建立一个目录

-- mkdir.cas

用 法 int mkdir (char *pathname);

相关函数

用 法 int rmdir (char * pathname);

原 型 在 dir.h

说 明 mkdir接受给定的pathname,建立一个给定名目录

rmdir删除由pathname给定的目录,该目录必须满足:

- 空
- 不是当前工作目录
- 不是根目录

返 回 值 当建立一新目录时, mkdir返回0。

在成功删除目录时, rmdir返回0。

在出现错误时,两个函数均返回-1,并置errno值为:

- | | |
|--------|-----------|
| EACCES | 无此权限 |
| ENOENT | 路径或文件名没找到 |

参 见 chdir

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <io.h>
int mkdir(const char *pathP)
{
    pushDS
asm      mov      ah, 039h
asm      LDS      dx, pathP

```

```

asm      int      021H
        popDS_
asm      jc      _mkdirFailed
        return(0);
mkdirFailed:
        return _IOerror(_AX);
}

```

MK_FP 设置一个选摘针

用 法 `#include <dos.h>`
`void far * MK_FP(unsigned seg,unsigned off);`

原 型 在 `dos.h`

说 明 见 `FP_OFF`

函 数 名 `__mknname` · 构造格式为 `TMPXXXXX.$$$` 的文件名。-- `tmpnam.c`

用 法 `char * pascal __mknname(char *s, unsigned num);`

原型文件 `_stdio.h`

返 回 值 格式为 `TMPXXXXX.$$$` 的文件名。

源 程 序

```

#include <stdio.h>
#include <_stdio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
static char template[L_tmpnam];
unsigned int tmpnam;
char * pascal __mknname(char *s, unsigned num) {
/* If no buffer provided, use internal template */
if (s == NULL)
    s = template;
/* A temporary name is build as follows: TMPXXXXX.$$$ */
*s = '\0';
strcat(s, "TMP");
ultoa((unsigned long)num, s + 3, 10);
strcat(s, ".$$$");
return (s);
}

```

mktemp 建立一个唯一的文件名 -- `mktemp.c`

用 法 `char * mktemp (char * template);`

原 型 在 `dir.h`

说 明 `mktemp` 使用一个唯一的文件名来替换 `template`, 并返回 `template` 的地址。

`template` 应为带有 6 个结尾 `x` 并以空字符终结的字符串。这些 `x` 用唯一的字母点集合替换, 这样, 在新文件名中有两个字母, 一个点和三个后缀字母。

从 `AAAAAA` 开始, 新文件在赋值前, 先查看磁盘中的文件名, 以避免出现相同式的文件名。

如果 `template` 构造得很好, `mktemp` 返回 `template` 串的地址; 否则, 它不创建打开文件。

可移植性 适用于 UNIX 系统

源 程 序

```

#include <dir.h>
#include <string.h>
#include <io.h>

```

```

char *mktemp(char *temp)
{
    register char *cp;
    int len;
    int i, j, k, l, m;
    len = strlen(temp);
    if (len < 6)
        return(0);
    cp = temp + len - 6;
    if (strcmp(cp, "XXXXXX") != 0)
        return(0);
    cp[2] = '?';
    for (i = 'A'; i <= 'Z'; i++)
    {
        cp[0] = i;
        for (j = 'A'; j <= 'Z'; j++)
        {
            cp[1] = j;
            for (k = 'A'; k <= 'Z'; k++)
            {
                cp[3] = k;
                for (l = 'A'; l <= 'Z'; l++)
                {
                    cp[4] = l;
                    for (m = 'A'; m <= 'Z'; m++)
                    {
                        cp[5] = m;
                        if (access(temp, 0777) == -1)
                            return(temp);
                    }
                }
            }
        }
    }
    return(0);
}

```

modf 把数分为指数和尾数

-- modf.cas

用 法 double modf (double value, double *iptr);

原 型 在 math.h

说 明 见fmod

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#pragma warn -rvl
double modf (double value, double *wholeP) {
    #if LDATA
    asm    push    ES
    #endif
    asm    FLD     DOUBLE (value)
    asm    mov     ax, value [6]
    asm    shl     ax, 1
    asm    cmp     ax, 0FFE0h
    /* infinite exponent ? */
    asm    jnb     mdf infinite
    asm    FLD     st(0)
    /* duplicate ST */
    asm    push    ax
    /* make a word on the stack */
    asm    mov     bx, sp
    asm    FSTCW   W0 (SS [bx])
    /* read out the current control word */
    asm    mov     ax, 0F3FFh
    asm    FWAIT
    asm    and     ax, SS [bx]
    /* mask out the rounding control */
    asm    or      ah, 0Ch
    /* chop towards zero */
    asm    push    ax
    asm    FLDCW   W0 (SS [bx-2])
    asm    pop     ax
    asm    FRNDINT
    /* round to integer */
    asm    FLDCW   W0 (SS [bx])
    /* restore original rounding control */
    asm    pop     ax
    asm    LES     bx, wholeP
    #endif
}

```



```

asm      FST      DOUBLE (ES_ [bx])      /* *wholeP = chop (value) */
asm      FSUBP    st(1), st              /* fraction = value - chop(value) */
mdf_end:
#ifdef LDATA
asm      pop      ES
#endif
return;
mdf_infinite:                          /* infinity == rounded (infinity) */
asm      LES      bx, wholeP
asm      FSTP     DOUBLE (ES_ [bx])
asm      FLDZ
asm      jmp      short mdf_end          /* zero = infinity - infinity */
}
#pragma warn .rvl

```

movedata 拷贝字节 -- movedata.cas

用 法 void movedata(int segsrc, int offsrc, int segdest, int offdest,
unsigned numbytes);

原 型 在 mem.h, string.h

说 明 movedata从源地址(segsrc:offsrc)拷贝numbytes字节到目标地址(segdest:offdest)中。

在微型、小型和中型模式程序中，由于数据段的地址没有显示给出，movedata对于移动远(far)数据是非常有用的。

memcpy可以用在紧缩、大型和巨型模式程序，这里的段地址是隐式给出的。

返 回 值 无

参 见 FP_OFF, memcpy, segread

源 程 序

```

#pragma inline
#include <mem.h>
#include <dos.h>
void movedata(unsigned srcseg, unsigned srcoff,
               unsigned dstseg, unsigned dstoff, size_t n) {
asm      cld
asm      mov      cx, n
asm      mov      di, dstoff
asm      mov      es, dstseg
asm      mov      si, srcoff
asm      push     ds
asm      mov      ds, srcseg
asm      shr      cx, 1
asm      rep      movsw
asm      jnc      mvd_end
asm      movsb
mvd_end:
asm      pop      ds
}

```

moverel 将当前位置(cp)移动一相对距离

用 法 #include <graphics.h>
void far moverel(int dx, int dy);

原 型 在 graphics.h

说 明 见 moveto

movetext 将屏幕文本从一个矩形区域拷贝到另一个矩形区域 -- movetext.c

用 法 int movetext(int left, int top, int right, int bottom,
int newleft, int newtop);

原型在 conio.h

说明 movetext将屏幕上由left、top、right和bottom定义的矩形中的内容拷贝到同样大小的新矩形中，新矩形的左上角为位置（newleft,newtop）。

所有的坐标均为绝对屏幕坐标

返回值 如果操作成功，movetext返回1；如果操作失败（如所给的坐标超过当前屏幕式的范围，movetext返回0。

模

可移植性 这些文本模式函数适用于IBM PC及与BIOS兼容的系统。

参 见 gettext

源 程 序

```
#include < video.h>
#include < conio.h>
int movetext(int sx1, int sy1, int sx2, int sy2, int dx1, int dy1)
{
    int first, last, direction, y;
    if (!__validatexy(sx1,sy1,sx2,sy2) || !__validatexy(dx1,dy1,dx1+(sx2-sx1),dy1+(sy2-sy1)))
        return 0;
    first = sy1;
    last = sy2;
    direction = 1;
    if (sy1 < dy1)
    {
        first = sy2;
        last = sy1;
        direction = -1;
    }
    for (y = first; y != last + direction; y += direction)
        __screenio(__vptr(dx1,dy1+(y-sy1)), __vptr(sx1,y), sx2-sx1+1);
    return 1;
}
```

moveto 将CP移到(x,y)

-- screen.c

用 法 #include <graphics.h>

void far moveto (int x,int y);

相关函数

用 法 void far moverel(int dx, int dy);

原 型 在 graphics.h

说 明 这些“移动当前位置”函数将CP移到屏幕的另一位置。

moveto将当前位置(CP)移到视区位置(x,y)。

将当前位置(CP)在x方向上移动dx象素，在y方向上移动dy象素。

返回值 无

可移植性 在Turbo Pascal 5.0中有相似的子程序

源 程 序

```
#include < video.h>
#include < dos.h>
#include < conio.h>
static void near pascal moveto(unsigned *newpos, unsigned *oldpos)
{
    DX = *newpos;
    if (DX != *oldpos)
    {
        BH = 0;
        AH = 2;
        _VideoInt();
        *oldpos = DX;
    }
    DL++;
    if (DL >= video.screenwidth)
    {
        DH++;
        DL = 0;
    }
}
```

```

    *newpos = _DX;
}

```

movmem 移动一字节块

-- movmem.cas

用法 void movmem(void *source, void *destin, unsigned len);

相关函数

用法 void setmem(void *addr, int len, char value); 原型在 mem.h

说明 movmem把len字节的块从source拷贝到destin。如果源和目标字符串重叠, 则选择拷贝方向, 使得数据正确地被拷贝。

源程序

```

#pragma inline
#include <asmrules.h>
#include <mem.h>
void movmem(void *src, void *dst, unsigned len) {
    #if LDATA
    #if !defined(__HUGE__)
        asm push ds
    #endif
    if (((void huge *)src) < ((void huge *)dst))
    #else
        ES = DS;
        if (src < dst)
    #endif
    {
        /* travel backward, need to adjust ptrs later */
        asm std
        asm mov ax, 1
    }
    else
    {
        /* travel forward, no need to adjust ptrs */
        asm cld
        asm xor ax, ax
    }
    asm LDS si, src
    asm LES di, dst
    asm mov cx, len
    asm or ax, ax
    asm jz movit
    asm add si, cx /* backward move, adjust ptrs to end-1 */
    asm dec si
    asm add di, cx
    asm dec di
    movit:
    asm test di, 1
    asm jz isAligned
    asm jcxz done
    asm movsb
    asm dec cx
    isAligned:
    asm sub si, ax /* compensate for word moves */
    asm sub di, ax
    asm shr cx, 1
    asm rep movsw
    asm jnc noOdd
    asm add si, ax /* compensate for final byte */
    asm add di, ax
    asm movsb
    noOdd:
    done:
    asm cld
    #if defined(__LARGE__) || defined(__COMPACT__)
        asm pop ds
    #endif
}

```

normalize 规格化指针. -- fbrk.c

用 法 局限于本模块

说 明 规格化一个远指针。

返 回 值 成功: 1;

失败: 0。

源 程 序

```
#include <alloc.h>
#include <dos.h>
#include <heap.h>
#define addressat(segment) ((void far*)((unsigned long)(segment) < 16))
static int near pascal normalize(char far *cp)
{
    extern unsigned _psp;
    static unsigned xsize = 0;
    unsigned size, result;
    size = 1 + FP_SEG(cp);
    size -= _psp;
    size = (size + 63) >> 6; /* Convert size to 1K units */
    if (size == xsize) /* Same as current size, do nothing */
    {
        _brklvl = cp;
        return(1);
    }
    size <= 6;
    result = FP_SEG(_heaptop);
    if (size + _psp > result)
        size = result - _psp;
    result = setblock(_psp, size);
    if (result == 0xffff)
    {
        xsize = size >> 6;
        _brklvl = cp;
        return(1);
    }
    else {
        _heaptop = addressat(_psp + result);
        /* If we fail, pass that back to the outer routines.
           This only happens if we did an exec of a program that
           did a keep. Also, adjust the _heaptop to make future calls
           faster. */
        return(0);
    }
}
```

normvideo 选择正常亮度字符

-- color.c

用 法 void normvideo (void);

原 型 在 conio.h

说 明 见highvideo

源 程 序

```
#include <video.h>
#include <conio.h>
#define INTENSE 0x08
void normvideo(void)
{ _video.attribute = _video.normattr;
}
```

nosound 关闭PC扬声器

-- sound.cas

用 法 void nosound (void);

原 型 在 dos.h

说 明 见sound

源程序

```
#pragma inline
#include <dos.h>
void nosound(void)
/* Turns the speaker off */
{
    outportb(0x61, inportb(0x61) & 0xfc);
} /* nosound */
```

_open 打开一个文件用于读或写

-- opena.cas

用法 #include <fcntl.h>

int _open (char *pathname,int access);

原型在 io.h

说明 见open

源程序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <fcntl.h>
#include <io.h>
int _open (const char *filename, int oflag)
{
    register int fd;
asm
    mov     al, 1
asm
    mov     cx, oflag
asm
    test    cx, O_WRONLY
asm
    jnz     ready
asm
    mov     al, 2
asm
    test    cx, O_RDWR
asm
    jnz     ready
asm
    mov     al, 0
ready:
    pushDS
    LDS     dx, filename
asm
    mov     cl, 0F0h
asm
    and     cl, oflag
asm
    or      al, cl
asm
    mov     ah, 3Dh
asm
    int     21h /* AX = sopen (DS:DX, AL) */
    popDS
asm
    jc      _openFailed
asm
    mov     fd,ax
    _openfd[fd] = (oflag & ~_O_RUNFLAGS) | O_BINARY;
    return(fd);
_openFailed:
    return _IOerror (_AX);
}
```

open 打开一个文件用于读或写

-- open.cas

用法 #include <fcntl.h>

#include <sys\stat.h>

int open (char *pathname,int access[, int permiss]);

相关函数 int _open(char *pathname,int access);

用法 int sopen(char *pathname,int access,int shflag,int permiss);

原型在 io.h

说明 open打开由pathname指定的文件，然后根据access的值进行读或写。

对于open来说，access是由以下两列表中的标志进行位“或”而构成的。第一个表中，只能使用一个标志。第二个中的标志可以任意逻辑组合使用。

表1: 读/写标志

O_RDONLY 打开用于只读
O_WRONLY 打开用于只写
O_RDWR 打开用于读写

表2: 其它存取标志

O_NDELAY 不用; 用于与UNIX兼容
O_APPEND 若置位, 每次写操作前都使文件指针量为文件末尾。
O_CREAT 如果文件已存在, 本标志无作用; 如果文件不存在, 则创建文件, 用 **perm** 位设置文件属性位, 象 **chmod** 一样
O_TRUNC 将存在文件的长度截为 0, 属性不变
O_EXCL 只和 **O_CREAT** 一起使用, 如果文件已存在, 返回错误。
O_BINARY 本标志显式表示文件以二进制方式打开
O_TEXT 本标志显式表示文件以文本方式打开

如果没有给出 **O_BINARY** 或 **O_TEXT**, 则文件按全局变量 **fmode** 设置的传送方式打开。
 如果使用了 **O_CREAT** 来构造 **access**, 则需要用 **sys/stat.h** 中定义的下列符号常量来提供 **open** 的参数 **perm**:

perm 值	存取权限
S_IWRITE	写允许
S_IREAD	读允许
S_IREAD S_IWRITE	读/写允许

对于 **_open**, MS-DOS 2.X 下, **access** 的值只限于 **O_RDONLY**, **O_WRONLY** 和 **O_RDWR**; 在 MS-DOS 3.X 下, 还可以使用下列值:

O_NOINHERIT	当文件没有传送给子程序时包含
O_DENYALL	只允许当前程序存取文件
O_DENYWRITE	只允许读其它打开的文件
O_DENYREAD	只允许写其它打开的文件
O_DENYNONE	允许其它程序共享打开的文件

在 DOS 3.X 下的一个 **_open** 调用中, 只允许包含一个 **O_DENYxxx** 值。这些文件共享属性在文件锁上时是例外的。

同时能打开的最大文件数由系统配置文件参数给出。

sopen 是以如下方式定义的宏:

open (pathname, (access|shflag), perm)

其中 **access**, **pathname** 和 **perm** 与 **open** 里一样, **shflag** 是一指明文件 **pathname** 共享属性的类型标志。 **shflag** 使用的符号常量在 **share.h** 中定义。

返回值 在完成成功后, 这些例程都返回一非零整数(为文件句柄)。文件指针(标明文件的当前位置)被置为开始处。在出错时, 它们都返回 -1, 并置 **errno** 为下列值之一:

ENOENT	路径或文件名没有找到
EMFILE	打开文件太多
EACCES	无此存取权限
EINVA	无效存取码

可移植性 **open** 和 **sopen** 适用于 UNIX 系统。在 UNIX 版本 7 中, 没有定义 **O_type** 助记符。UNIX 系统 III 使用除 **O_BINARY** 外的所有 **O_type** 助记符。

_open 只适用于 MS-DOS

参 见 **chmod**, **close**, **creat**, **dup**, **ferror**, **fmode**(变量), **fopen**, **lock**, **lseek**, **read**, **searchpath**, **setmode**, **write**
源 程 序

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
#include <fcntl.h>
#include <sys/stat.h>
extern unsigned _notUmask;
int open(pathP, oflag, mode)
    const char *pathP; register int oflag; unsigned mode;
/* Open a new file or replace an existing file with the given pathname.
   The opened file's read/write permission will be (pmode && ! notUmask),
   unless the file already exists in which case its present mode is kept. */
{
    unsigned attrib;
    unsigned devstat;
    register int fildes;
    if (! (oflag & (O_TEXT | O_BINARY)))
        oflag |= _mode & (O_TEXT | O_BINARY);
    if (oflag & O_CREAT)
    {
        if (((mode & _notUmask) & (S_IREAD | S_IWRITE)) == 0)
            IOerror (1);
        if ((attrib = _chmod (pathP, 0)) == ~0U)
        {
            attrib = (mode & S_IWRITE) ? 0 : 1 /* read-only */;
        }
        else
        {
            if (oflag & O_EXCL)
                return IOerror (e fileExists);
            else /* ignore O_CREAT if file exists */
                goto OpenExisting;
        }
    }
    /* Are any sharing/inheritance bits specified? If so, then we need to use
       open(), so we must create, close, then reopen it. Since the creation
       may be read only mode, but we may need read-write access, we must create
       it with write access to allow us to reopen for write, and then later
       change the mode after a successful open. */
    if (oflag & 0xF0) /* any sharing/inheritance? */
    {
        if ((fildes = dosCreat ((char *)pathP, 0)) < 0)
            return fildes;
        close (fildes);
        goto OpenWithSharing;
    }
    else
        if ((fildes = dosCreat ((char *)pathP, attrib)) < 0)
            return fildes;
    }
    else
    {
OpenExisting:
        attrib = 0; /* inhibit _chmod, see below */
OpenWithSharing:
        if ((fildes = _open (pathP, oflag)) >= 0)
        {
            if ( (devstat = ioctl (fildes, 0)) & 0x80)
            {
                oflag |= O_DEVICE;
                if (oflag & O_BINARY)
                    ioctl(fildes, 1, (void *)((devstat & 0xFF) | 0x20));
            }
            else
            {
                if ((oflag & O_TRUNC))
                    dosWriteNone (fildes);
                if (! (oflag & (O_BINARY | O_RDONLY)))
                    _TrimCrlf (fildes);
            }
        }
        /* set shared file to read-only */
        if (attrib && oflag & 0xF0)
        {
            _chmod (pathP, 1, 1);
        }
    }
}

```

```

    }
    if (fildes >= 0)
    {
        _openfd [fildes] = (oflag & ~ O_RUNFLAGS) |
            ((oflag & (O_CREAT | O_TRUNC)) ? O_CHANGED : 0);
    }
    return fildes;
}

```

_openfp 打开文件。 -- fopen.c

用 法 static FILE * pascal near _openfp (FILE *fp,
const char *filename, const char *type);

原型文件 局限于本模块

说 明 打开一个文件。

返 回 值 成功: 每个函数返回新打开的文件流。freopen返回参数流。

失败: 返回NULL。

源 程 序

```

static FILE *pascal near _openfp (FILE *fp, const char *filename,
const char *type)
{
    unsigned oflag, mode;
    if (((fp->flags = CheckOpenType (type, &oflag, &mode)) == 0) ||
        ((fp->fd < 0) &&
         (fp->fd = open (filename, oflag, mode)) < 0))
    {
        fp->fd = -1;
        fp->flags = 0;
        return NULL;
    }
    if (isatty (fp->fd))
        fp->flags |= F_TERM;
    if (setvbuf (fp, NULL, (fp->flags & _F_TERM) ? _IOLBF : _IOFBF, BUFSIZ))
    {
        fclose (fp);
        return NULL;
    }
    else { fp->istemp = 0;
           return fp;
         }
}

```

outp 输出整数到硬件端口中

用 法 void outp (int port,int value);

原 型 在 dos.h

说 明 见inport

output 输出整数到硬件端口中

-- output.cas

用 法 void output (int port,int value);

原 型 在 dos.h

说 明 见inport

源 程 序

```

#pragma inline
#include <dos.h>
#undef outputb
void output(int port, int val) {
asm    mov    dx, port
asm    mov    ax, val
asm    out    dx, ax
}

```


outportb 输出字节到硬件端口中 - outport.cas

用 法 `#include <dos.h>`
`void outportb (int port,char byte);`

原 型 在 `dos.h`

说 明 见 `inport`

源 程 序

```
void outportb(int port, unsigned char val)
{
    asm      mov     dx, port
    asm      mov     al, val
    asm      out     dx, al
}
```

outtext 在视区中显示一个字符串

用 法 `#include <graphics.h>`
`void far outtext (char far *textstring);`

相关函数

用 法 `void far outtextxy (int x,int y,char far *textstring);`

原 型 在 `graphics.h`

说 明 这两个函数使用当前对齐方式、当前字体、方向和大小在视区中显示一文本字符串。

`outtext` 在 CP 处输出 `textstring`。如果水平文本对齐方式是 `LEFT_TEXT`，并且文本方向是 `HORIZ_DIR`，则 CP 的 x 坐标将增大 `textwidth(textstring)`；否则 CP 保持不变。

`outtextxy` 在给定位置(x,y)处输出 `textstring`。

为了在使用几种字体时保持代码的兼容性，请使用 `textwidth` 和 `textheight` 函数决定字符串的尺寸大小。

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 `gettextsettings, textheight`

outtextxy 将一个字符串送到指定位置

用 法 `#include <graphics.h>`
`void far outtextxy (int x,int y, char *textstring);`

说 明 见 `outtext`

parsfnm 分析文件名 - parsfnm.cas

用 法 `#include <dos.h>`
`char *parsfnm (char *cmdline,struct fcb *fcbptr,int option);`

原 型 在 `dos.h`

说 明 `parsfnm` 分析 `cmdline` 所指的字符串，通常为命令行，以找到一文件名。该文件名(包括驱动器，文件名和扩展名)被放在由 `fcbptr` 所指的 FCB 中。

`option` 参数是 DOS 分析系统调用中 AL 说明的值。对于文件名进行分析详细描述，可参考《MS-DOS 程序员参考手册》中有关系统调用 0x29 部分的说明。

返 回 值 在成功分析一个文件名后，`parsfnm` 返回指向紧跟文件名后的字节的指针。如果在分析过程中出错，`parsfnm` 返回 0。

可移植性 只适用于 MS-DOS

源 程 序

`#pragma inline`

```

#include <asmrules.h>
#include <dos.h>
#include <stddef.h>
char *parsfnm(const char *cmdline, struct fcb *fcb, int opt)
#ifdef (LDATA)
    _ES = _DS;
#endif
    pushDS
    LDS    _SI, cmdline
    LES    _DI, fcb
    asm    mov    al, opt
    asm    mov    ah, 29h
    asm    int    21h
    popDS
    asm    cmp    al, 0ffh
    asm    jne    parsret
    return NULL;
parsret:
#ifdef (LDATA)
    asm    mov    es, W1(cmdline)
    return (char _es *) _SI;
#else
    return (char *) _SI;
#endif
}

```

peek 检查存储单元

-- peek.c

用 法 int peek (int segment,unsigned offset);

相关函数

用 法 char peekb (int segment,unsigned offset);

原 型 在 dos.h

说 明 peek 和 peekb 检查由 segment:offset 指定的存储单元。

如果在调用这些例程时包含了 dos.h 文件, 则它们被当做宏对待而扩展为插入代码。如果没有包含 dos.h(或者虽包含了, 但使用了#undef 例程指令), 则得到函数而不是宏。

返 回 值 peek 和 peekb 返回存储地址 segment:offset 中的值。peek 返回一字, 而 peekb 返回一字节。

可移植性 只适用于 8086 系列。

参 见 harderr, poke

源 程 序

```

#include <dos.h>
#ifdef peek #undef peekb
int peek(unsigned segment, unsigned offset)
{
    _ES = segment;
    return(* (int _es *) offset);
}

```

peekb 检查存储单元

-- peek.c

用 法 #include <dos.h>

char peekb (int segment,unsigned offset);

原 型 在 dos.h

说 明 见 peek

源 程 序

```

char peekb(unsigned segment, unsigned offset)
{
    _ES = segment;
    return(* (unsigned char _es *) offset);
}

```

用 法 void perror (char *string);

原 型 在 stdio.h

说 明 perror 打印错误信息到 stderr 中，表示当前程序中系统调用最近遇到的错误。

首先打印参数 string，接着是冒号，然后是相对于当前 errno 值的信息，最后是换行。约定是传送程序名为参数字符串。

为了控制信息格式，信息字符串数组在 sys_errlist 中提供。errno 可以作为数组的下标以查找对应错误号的字符串。该字符串中不包括换行符。

sys_nerr 包含数组中的入口数。

对于更详细的信息，可参阅有关 errno 和 sys_errlist 的说明。

返 回 值 无

可移植性 适用于 UNIX 系统。

参 见 eof

源 程 序

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
char *sys_errlist[] = {
    "Error 0",
    "Invalid function number",
    "No such file or directory",
    "Path not found",
    "Too many open files",
    "Permission denied",
    "Bad file number",
    "Memory arena trashed",
    "Not enough memory",
    "Invalid memory block address",
    "Invalid environment",
    "Invalid format",
    "Invalid access code",
    "Invalid data",
    0,
    "No such device",
    "Attempted to remove current directory",
    "Not same device",
    "No more files",
    "Invalid argument",
    "Arg list too big",
    "Exec format error",
    "Cross-device link",
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    "Math argument",
    "Result too large",
    "File already exists"
};
int sys_nerr = sizeof sys_errlist / sizeof sys_errlist[0];
void perror(const char *s)
{
    char *cp;
    if (errno < sys_nerr && errno >= 0)
        cp = sys_errlist[errno];
    else
        cp = "Unknown error";
    fprintf(stderr, "%s: %s\n", s, cp);
}
```

pieslice 画并填充扇形

用 法 #include <graphics.h>

```
void far pieslice (int x,int y,int stangle,
                  int endangle,int radius);
```

原型在 graphics.h

说明 见 arc

poke 存值到一个给定存储单元 -- poke.c

用法 void poke (int segment ,int offset,int value);

相关函数

用法 void pokeb(int segment,int offset,char value);

原型在 dos.h

说明 poke 把整数 value 存到存储单元 segment:offset 处。

在调用这些例程时，如果包含了 dos.h，它们将作为可扩展为插入代码的宏看待；如果没有包含 dos.h(或虽然包含了，但使用了#undef 例程)，将得到函数而不是宏。

pokeb 同 poke 相近，只是它存的是一个字节 value 而不是一个整数。

返回值 无

可移植性 只适用于 8086 系列

参见 peek

源程序

```
#include <dos.h>
#ifdef poke
#ifdef pokeb
void poke(unsigned segment, unsigned offset, int value)
{
    ES = segment;
    * (int _es *) offset = value;
}
```

pokeb 存值到一个给定存储单元 -- poke.c

用法 #include <dos.h>

void pokeb(int segment,int offset,char value);

原型在 dos.h

说明 见 poke

源程序

```
void pokeb(unsigned segment, unsigned offset, char value)
{
    ES = segment;
    * (char _es *) offset = value;
}
```

poly 根据参数产生一个多项式 -- poly.cas

用法 double poly(double x ,int n, double c[]);

原型在 math.h

说明 poly 产生一个 n 次的 x 多项式，系数为 c[0], c[1],...,c[n]。如 n=4 产生的多项式为：

$$c[4]x^4 + c[3]x^3 + c[2]x^2 + c[1]x + c[0]$$

返回值 poly 返回给定 x 的多项式值。

可移植性 适用于 UNIX 系统

源程序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
```

```

#include < math.h>
#include <errno.h>
#pragma warn -rvl
#pragma warn -use
double poly (double x, int n, double c [])
{
    unsigned sw;
    register SI, DI; /* prevent the compiler making its own usage */
    asm FLD DOUBLE (x)
    asm mov si, n
    asm mov cl, 3
    asm sbl si, cl
    asm or si, si
    asm jl ply_domain
    asm LES bx, c
    asm FLD DOUBLE (ES [bx+si])
    asm jz short ply_end
ply_loop:
    asm FMUL ST, ST(1)
    asm sub si, 8
    asm FADD DOUBLE (ES [bx+si])
    asm jg ply_loop
ply_end:
    asm FXAM
    asm FSTSW W0 (sw)
    asm FSTP ST(1) /* discard ST(1) */
    asm mov ax, sw
    asm sahf
    asm jc ply_range
    return;
ply_domain:
    asm mov si, DOMAIN
    asm jmp short ply_err
ply_range:
    asm mov si, OVERFLOW
ply_err:
    asm FSTP ST(0) /* discard ST */
#pragma warn -ret
    return matherr (_SI, "poly", &x, c, HUGE_VAL);
#pragma warn .ret
}
#pragma warn .rvl
#pragma warn .use

```

pow 指数函数 x

-- pow.cas

用 法 double pow(double x,double y);

原 型 在 math.h

说 明 见 exp

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include < math.h>
#include <errno.h>
#pragma warn -rvl
#pragma warn -pow
#pragma warn -use
double pow (double x, double y)
{
    double temp; /* also used as a 64-bit integer */
    unsigned sw_y; /* status-word from testing y */
    char negate = 0; /* boolean, negate after exp() ? */
    register SI; /* prevent the compiler using SI */
    asm FLD DOUBLE (x)
    asm mov bx, 7FF0h /* mask just the exponent */
    asm mov ax, x [6]
    asm and ax, bx

```

```

asm    jz     pow_ofZero
asm    cmp    ax, bx
asm    je     pow_ofInfinity
asm    FLD    DOUBLE (y)
asm    mov    ax, y [6]
asm    and    ax, bx
asm    jnz    temp1
asm    jmp    pow_toZero
temp1:
asm    cmp    ax, bx
asm    je     pow_toInfinity
asm    jmp    pow_normal
/****
/* Raising any number to infinity is treated as a range error. */
pow_toInfinity:
asm    FSTP   DOUBLE (temp)          /* propagate Y thru to result */
asm    jmp    short pow_discard
/* Powers of infinity are range errors. */
pow_ofInfinity:
asm    FSTP   DOUBLE (temp)          /* propagate X thru to result */
asm    mov    ax, y[6]
asm    or     ax, ax
asm    jge    pow_overflow            /* jump if exponent nonnegative */
asm    mov    si, UNDERFLOW
asm    temp = 0.0;
asm    jmp    short pow_complain
pow_discard:
asm    FSTP   ST(0)                  /* discard X */
pow_overflow:
asm    mov    si, OVERFLOW
asm    jmp    short pow_complain

/* Powers of 0 are (EDOM, 1, 0) as Y ranges over (negative, zero, positive). */
pow_ofZero:
asm    FSTP   ST(0)                  /* discard X */
asm    mov    ax, y [6]
asm    or     ax, ax                  /* was Y positive ? */
asm    jg     pow_zero
asm    mov    si, DOMAIN
asm    je     pow_zz
asm    temp = HUGE_VAL;
asm    jmp    short pow_complain
pow_zz:
asm    temp = 1.0;
pow_complain:
asm    return _matherr (_SI, "pow", &x, &y, temp);
/* return zero. */
pow_zero:
asm    FLDZ
asm    return;
/* Arrive here if Y is zero. The zero'th power of any number is 1. */
pow_toZero:
asm    FSTP   ST(0)                  /* discard Y */
asm    FSTP   ST(0)                  /* discard X */
/* return one. */
asm    FLD1
asm    return;
/**** End of Special Cases ****/
/* If arrived here then both x and y seem to be ordinary numbers. */
pow_normal:
asm    FCLEX
asm    FRNDINT
asm    FSTSW  W0 (sw_y)              /* is Y an integer */
asm    FWAIT
asm    test   BY0 (sw_y), 20h         /* precision error if not */
asm    jz     pow_integral
asm    FSTP   ST(0)                  /* discard Y */
/* Arrive here if the exponent exceeds integer range or if it contains
a fractional part. Calculate using Log and Exp functions. Just

```

```

x is on 87-stack. */
pow_fractional:
asm sub sp, 8
asm mov bx, sp
asm fstp double (ss [bx])
asm call extproc (log)
/* arg is > 0, so log cannot fail */
asm fmul double (y)
asm mov bx, sp
asm fstp double (ss [bx])
asm call extproc (exp)
    if (negate)
    {
        asm fchs
    }
asm add sp, 8
return;
/* If arrived here then Y is some integer of up to 64 bits and has
   been copied to temp. Y is ST(0), X is ST(1), AX is exponent of Y. */
pow_integral:
asm mov cl, 4
asm ror ax, cl
asm sub ax, 3FFh /* remove the bias */
asm cmp ax, 63 /* AX = n, the exponent */
asm jb pow_trueIntegral /* discard Y */
asm fstp ST(0)
asm jmp short pow_fractional
/* The shift-and-add method is not accurate for extreme powers since
   round off errors are magnified. However, we cannot simply call for
   evaluation like fractional powers because X may be negative and
   fractional negative powers are treated as exceptions. */
pow_trueIntegral:
asm cmp al, 12
asm jb pow_shiftAndAdd
pow_unsafeRange:
asm fistp qword ptr (temp) /* store an integer copy of Y */
asm test BY0 (x [7]), 80h /* X less than 0 ? */
asm jz pow_fractional /* X not signed, so no worry */
asm fchs /* make X absolute */
asm test BY0 (temp), 01h /* odd or even ? */
asm jz pow_fractional /* even powers are positive */
/* If we arrive here then X was negative and Y was odd. Calculate with
   abs(X) and then negate result.*/
negate = 1;
asm jmp short pow_fractional
/* Arrive here for modest integral powers of any number. We must also
   check for overflow, by making a worst-case check on log (X^Y). If
   it has a potential to overflow, then we use the exp(log()) method.*/
pow_shiftAndAdd:
asm mov bx, x [6]
asm shl bx, 1
asm sub bx, 7FE0h /* BX estimates log2 (X) */
asm mov dx, bx
asm xchg cx, ax
asm inc cx /* 2^CL is max possible Y */
asm shl bx, cl /* multiply BX by max Y */
asm sar bx, cl
asm dec cx
asm xchg ax, cx
asm cmp bx, dx /* did BX lose any bits ? */
asm jne pow_unsafeRange
asm fld ST (1) /* Z = X */
asm mov dx, y [4]
asm mov bl, y [6]
asm and bl, 0Fh /* most significant nibble */
asm and dl, 0F0h /* DX is the next 12 bits */
asm or dl, bl
asm ror dx, cl /* top 16 bits of fraction */
pow_iWhileBit:
asm dec al

```

```

asm    jl     pow_maybeInverse          /* Z *= Z    */
asm    FMUL   ST(0), ST(0)
asm    shl    dx, 1
asm    jnc    pow_iWhileBit
asm    FMUL   ST(0), ST(2)              /* Z *= X    */
asm    jmp     short    pow_iWhileBit
pow_maybeInverse:
asm    FSTP   ST(1)                    /* overwrite Y */
asm    test   BY0 (& F7), 80h         /* was Y a negative power ? */
asm    FSTP   ST(1)                    /* overwrite X */
asm    ja     pow_iDone
asm    FLDI
asm    FDFVFP ST(1), ST(0)             /* if so, invert result. */
pow_iDone:
    return;
}

```

pow10 指数函数 10

— pow10.cas

用法 double pow10(int p);

原型在 math.h

说明 见 exp

源程序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <math.h>
#include <errno.h>
#include <stddef.h>
typedef unsigned short int    extend [5]; /* 80-bit constants */
static const float e0to7 [8] =
{ 1, 1.e1, 1.e2, 1.e3, 1.e4, 1.e5, 1.e6, 1.e7,
};
/* Exponents > 4932 become infinities. Exponents < -4932 become 0. */
static const float e8 = 1.e8;
static const double e16 = 1.e16;
static const extend e32 = {0xB59E, 0x2B70, 0xADAA, 0x9DC5, 0x4069};
static const extend e64 = {0xA6D5, 0xFFCF, 0x1F49, 0xC278, 0x40D3};
static const extend e128 = {0x8CDF, 0x80E9, 0x47C9, 0x93BA, 0x41A8};
static const extend e256 = {0xDE8C, 0x9DF9, 0xEBFB, 0xAA7E, 0x4351};
static const extend e512 = {0x91C7, 0xA60E, 0xA0AE, 0xE319, 0x46A3};
static const extend e1024 = {0x0C17, 0x8175, 0x7586, 0xC976, 0x4D48};
static const extend e2048 = {0x5DE5, 0xC53D, 0x3B5D, 0x9E8B, 0x5A92};
static const extend e4096 = {0x979B, 0x8A20, 0x5202, 0xC460, 0x7525};
static const float eINF = 1.0/0.0;
#pragma warn -rvl
double pow10 (int p) {
#define MAX_87_EXP 4932
#ifdef _HUGE_
asm    mov     ax, seg e0to7
asm    mov     DS, ax
#endif
/* Take care of all the easy special cases up front. */
asm    mov     ax, p
asm    if ((int)_AX < -MAX_87_EXP) /* Extremely small -> Zero */
asm    {
asm    FLDZ
asm    jmp     p10_end
asm    }
asm    if ((int)_AX > MAX_87_EXP) /* Extremely large -> Infinity */
asm    {
asm    FLD     FLOAT(eINF)
asm    jmp     p10_end
asm    }
asm    if ((int)_AX == 0) /* 10^0 -> 1.0 */
asm    {

```



```

asm      FLD1
asm      jmp      p10_end
        }
/*-- The non-trivial cases require some calculation. --*/
/*asm    mov      ax, p*/
asm      or       ax, ax
asm      jnl      p10_abs
asm      neg      ax
p10_abs:
asm      mov      si, 7
asm      and      si, ax
asm      shl      si, 1
asm      shl      si, 1
asm      FLD      FLOAT (e0to7 [si])
asm      shr      ax, 1
asm      shr      ax, 1
asm      shr      ax, 1
p10_maybe8:
asm      shr      ax, 1
asm      jnc      p10_maybe16
asm      FMUL     FLOAT (e8)
p10_maybe16:
asm      jnz      keep_going
asm      jmp      p10_checkSign /* optimization, skip if all done */
keep_going:
asm      shr      ax, 1
asm      jnc      p10_maybe32
asm      FMUL     DOUBLE (e16)
p10_maybe32:
asm      shr      ax, 1
asm      jnc      p10_maybe64
asm      FLD      LONGDOUBLE (e32)
asm      FMUL
p10_maybe64:
asm      shr      ax, 1
asm      jnc      p10_maybe128
asm      FLD      LONGDOUBLE (e64)
asm      FMUL
p10_maybe128:
asm      shr      ax, 1
asm      jnc      p10_maybe256
asm      FLD      LONGDOUBLE (e128)
asm      FMUL
p10_maybe256:
asm      shr      ax, 1
asm      jnc      p10_maybe512
asm      FLD      LONGDOUBLE (e256)
asm      FMUL
p10_maybe512:
asm      shr      ax, 1
asm      jnc      p10_maybe1024
asm      FLD      LONGDOUBLE (e512)
asm      FMUL
p10_maybe1024:
asm      shr      ax, 1
asm      jnc      p10_maybe2048
asm      FLD      LONGDOUBLE (e1024)
asm      FMUL
p10_maybe2048:
asm      shr      ax, 1
asm      jnc      p10_maybe4096
asm      FLD      LONGDOUBLE (e2048)
asm      FMUL
p10_maybe4096:
asm      shr      ax, 1
asm      jnc      p10_checkSign
asm      FLD      LONGDOUBLE (e4096)
asm      FMUL
p10_checkSign:

```

```

asm    test    BY1(p), 80h
asm    jz      p10_end
/* 10^(-n) = 1 / 10^n, so we need the reciprocal of 10, */
asm    FIDIVR  FLOAT(e0to7)      /* TOS = 1.0 / TOS */
/* Now the value 10^p is on TOS. */
p10_end:
    return;
}
#pragma warn reset

```

...printf 产生格式化输出的函数 -- printf.c

用 法 int printf (char *format,...);

相关函数 int cprintf(char *format[,argument,...]);

用 法 int fprintf(FILE *stream, char *format[, argument, ...]);

int sprintf(char *string, char *format[, argument, ...]);

int vfprintf(FILE *stream, char *format, va_list param);

int vprintf(char *format, va_list param);

int vsprintf(char *string, char *format, va_list param);

原 型 在 stdio.h

说 明 ...printf 系列函数都产生格式化输出:

- 接受确定输出格式(由用法中的 format 给定)的格式字符串(format string)
- 将格式字符串传给个数变化的参数以产生格式化输出(值在 argument 或 va_list param 中给定)。

输出位置在以下三个函数中是隐式给出的:

printf 和 vprintf 把输出送到 stdout

cprintf 把输出直接送到控制台

其它四个...printf 函数还接受另外一个参数(参数表中第一个)。这个另外参数指定输出地点。

fprintf 和 vfprintf 把输出放到给定流中。

sprintf 和 vsprintf 把输出放到存储区的一个字符串中

四个...printf 函数从函数调用中接受要格式化的参数(printf, cprintf, fprintf 和 sprintf)。

其它三个函数(vprintf, vfprintf 和 vsprintf)从一个变化参数表中接受要格式化的参数。

v...printf 函数是...printf 函数的可选择入口点。

为得到更多的信息,参阅 va...的定义。

下面对...printf 函数作一小结:

printf 把输出送到 stdout

cprintf 直接把输出送到控制台

fprintf 把输出送到给定名流 stream

sprintf 把输出送到以空字符终结的字符串中,用户应确保 string 中有足够的空间存放字符串。

vprintf 同 printf 相似,只是它从 va_arg 数组的 va_list param 中接受参数

vfprintf 同 fprintf 相似,只是它从 va_arg 数组的 va_list param 中接受参数

vsprintf 同 sprintf 相似,只是它从 va_arg 数组的 va_list param 中接受参数

关于如何使用 vprintf 的例子,请见...printf

1. 格式字符串:

每个...printf 函数调用中的格式字符串用于控制函数转换方式、格式化和输出其参数。

对于格式,必须有足够的参数相对应,否则,结果是不可预测,也有可能是灾难性

的。过多的参数（超过格式所要求的）将被忽略。

格式字符串是包含两类目标—简单字符(plain characters)和转换指示(conversion specification)的字符串：

简单字符只是简单地将字符拷贝到输出流中。

转换指示从参数表中取参数，并对它们进行格式化。

2. 格式指示：

...printf 的格式指示有以下形式：

% [flags] [width] [.prec] [F|N|h|L] [type]

每一转换指示以百分号(%)开始，后面按顺序为：

- 可选的标志字符序列 [flags]
- 可选的宽度指示符 [width]
- 可选的精度指示符 [.prec]
- 可选的输入长度修改符 [F|N|h|L]
- 转换类型字符 [type]

3. 可选的格式字符串成份：

以下列出格式字符串中可选的字符、指示符和修改符所控制的格式输出的几个方面：

字符或指示符	控制或指示什么
flags	输出对齐、数值符号、小数点、尾零、八进制或十六进制数
width	输出字符和填补空格或零的最小数
precision	输出字符最大数，对于整型值，为输出最少数字个数
size	覆盖缺省的参数大小（N=近指针，F=远指针，h=短整型，l=长整型）

4. ...printf 转换类型字符：

列表列出...printf 的转换类型字符，所接受的输入参数类型和输出的格式。

类型字符表中的信息是根据假定没有标志字符、宽度指示符、精度指示符或输入大小修饰符的情况下给出的。为知道可选择字符和指示符是怎样影响...printf 的输出，请看下表：

类型字符	输入参数	输出格式
d	整数	带符号十进制整数
i	整数	带符号十进制整数
o	整数	无符号八进制整数
u	整数	无符号十进制整数
x	整数	无符号十六进制整数 (有a,b,c,d,e,f)
X	整数	无符号十六进制整数 (有A,B,C,D,E,F)
f	浮点数	格式为[-]dddd.dddd的带符号数值
e	浮点数	格式为[-]d.dddd e [+/-]ddd的带符号数值
g	浮点数	由给定值和精度确定的e或f格式的带符号值。尾部

零和小数点在必要时输出。

E 浮点数

同e相似，只是用E表示指数

G 浮点数

同g相似，只是使用e格式时，用E表示指数

		字符类
c	字符	单个字符
s	字符串	输出字符直到空字符终 结符或达到所要求的精度
%	无	输出%字符

		指针类
d	指向整数的指针	把目前已写字符的个数存到由输入 参数所指的位置中
p	指针	按指针输出输入参数 远指针被打印为: XXXX.YYYY 近指针被输出为: YYYY (只是偏移量)

5. 转换:

下表列出这些指示符的一些转换:

字符	转换
e或E	参数被转换为格式，其中: <ul style="list-style-type: none"> • 在小数点前有一位数字 • 小数点后的数字个数等于精度 • 指数总是包含三位数
f	参数被转换成十进制格式，其中: 小数点后的数字个数表示精度 (如果给定非零精度)
g或G	参数以e,E或f格式输出精度指明有效数字位，尾部的零结果中被删除，只有当必要的时候才出现小数点。 如果转换字符为g，参数以e或f格式 (带某些限制) 输出; 如果转换字符为G，参数以E格式输出。E格式只有当转换的结果使指数为下列值之一时才使用: <ul style="list-style-type: none"> <a> 大于精度 小于-4
x或X	对x转换，字母a,b,c,d,e,f可出现在输出中; 对于X转换，字母A,B,C,D,E和F可出现在输出中。

6. 标志字符:

标志字符为减号(-)，加号(+)，井号(#)和空格，它们可以以任意顺序和组合出现。

标志	所指示意义
----	-------

结果左对齐，右边填充格; 若不给定的话，结果右对齐，左边填充格或零。

+	带符号的转换, 结果总以加号或减号开头
空格	如果值为非负值, 输出以空格代替加号; 负值以减号开头。
#	指定参数以选择格式转换, 见下表。

注意: 如果同时给出加号和空格, 加号优先。

7. 选择格式

如果井号标志(#)和转换字符一起使用, 将对被转换的参数(arg)产生如下影响:

转换字符	#号影响
c,s,d,i,u	无影响
0	0被预置于非零参数arg前面
x或X	0x(0X)被预置于非零参数arg前面结果总是包含小数点, 即使其后没有数字
e、E或f	数字。一般情况下, 只有当小数点后有数字时, 才出现小数点。
g或G	同e和E相似, 只是不删除尾部零

8. 宽度指示符

宽度指示符设置输出值的最小字段宽度。

宽度用两种方法指定: 直接方法用十进制数字字符串或间接方法用一星号(*)。如果使用星号作为宽度指示符, 调用中的下个参数(必须为整数)指示最小输出字段宽度。没有任何情况会使得不存在或过小的字段宽在输出时被截断, 如果转换后的结果大于段宽, 则字段被扩展到能包含转换结果的长度。

宽度指示符	输出宽度影响
n	至少输出n个字符。如果结果值少于n个字符, 输出填空(在给定“-”标志时, 右边填空, 否则左边填空)
On	至少输出n个字符。如果结果值少于n个字符, 左边填零。
*	参数表提供指示宽度指示符, 在实际格式参数的前面。

9. 精度指示符

精度指示符总是以点(.)开头, 用以区别任何前导的宽度指示符。同宽度指示符一样, 精度指示符也可用直接方法—通过一十进制数字串或间接方法—通过一星号给出。如果使用星号(*)作为精度指示符, 调用中的下一参数(整型数)应该指明精度。如用星号来说明宽度或精度或两者, 则宽度参数必须紧跟在星号后, 然后是精度参数。

精度指示符	输出精度影响
	对于d,i,o,u,x,X类型, 缺省值=1
	对于e,E,f类型, 缺省值=6

	对于g,G类型, 缺省值=所有有效数字
	对于S类型, 缺省值=输出第一个空字符
	对于C类型, 无影响
.o	对于d,i,o,u,x类型, 精度为缺省值对于e,E,f类型, 不输出小数点
.n	输出n个字母或n个十进制位。如果输出多于n个字符, 输出被截断或舍入(根据类型字符而定)
.*	参数表提供精度指示, 必须在实际格式参数前给定。

转换字符	精度指示符(.n)对转换的影响
d,i	.n指明至少有n个数字要输出
o,u	当输入参数少于n个数字时, 输出左边填
x,X	零; 当输入参数多于n个数字时, 输出不被截断。
e,E	.n指明在小数点后有n个字符要输出, 最后一位数字被舍入。
g,G	.n指明最多输出n个有效数字
c	.n对输出无影响
s	.n指明输出不多于n个字符

10. 输入长度修饰符

输入长度修饰符(F,N,h或l)给定以下输入参数的长度:

F=远指针

N=近指针

h=短整型

l=长整型

输入长度修饰符(F,N,h,l)影响...printf函数对对应于输入参数的数据类型的解释。F和N只用于输入参数为指针(%p,%s,%n), h和l用于输入args为数字的情况(整数和浮点型)。

F和N都重新解释输入arg。通常, 对于%p,%s或%s转换, arg是存储模式的缺省大小指针。"F"表示"解释arg为远指针", "N"表示为"解释arg为近指针"。

h和l都覆盖数字输入参数的缺省大小: L用于整型(d,i,o,u,x,X)和浮点型(e,E,f,g和G),而h只用于整型。h和l对字符型(c,s)或指针型(p,n)均不产生影响。

输入长度修饰符	参数解释方式
F	参数为远指针
N	参数为近指针
h	参数为短整型(对d,i,o,u,x和X)
l	参数为长整型(对d,i,o,u,x和X)
	参数为双精度型(对e,E,f,g和G)

返回值 每一函数均返回输出的字节数。sprintf不包括空字符。在出错时, 返回EOF。

可移植性 函数printf, fprintf, sprintf和vsprintf适用于UNIX系统

在 Kernighan 和 Ritchie 书中有定义。

vprintf, vfprintf 和 vsprintf 适用于 UNIX 系统

在 Kernighan 和 Ritchie 书中没有定义。

参 见 ecvt, fread, putc, puts, scanf, va...

源 程 序

```
#include <stdio.h>
#include <_printf.h>
#include <_stdio.h>
int cdecl printf(const char *fmt, ...)
{
    return __vprinter ((putnF *)__fputn, stdout, fmt, _va_ptr);
}
```

putc 输出一字符到流中

用 法 #include <stdio.h>

int putc(int ch, FILE *stream);

相关函数 int fputc(int ch, FILE *stream);

用 法 int fputc(char ch);

int putchar(int ch);

int putchar (int ch);

int putw(int w, FILE *stream);

原 型 在 stdio.h

说 明 putc 是一把字符 ch 送到指定输出流的宏。

putchar(ch)是一定义为 putc(ch;stdout)的宏。

fputc 与 putc 类似, 只是它是输出 ch 到指定名流的函数。

fputc 输出 ch 到 stdout, fputc(char ch)等价于 fputc(char, stout)。

putch 输出字符 ch 到控制台。

putw 输出整数 w 到输出流 stream, 它不接受也不产生文件中的特有对齐。

返 回 值 成功时, putc, fputc, fputc 和 putchar 返回字符 ch, putw 返回整数 w, putch 什么也不返回。

在错误发生时, 除 putch 外的所有函数都返回 EOF, putch 什么也不返回。

因为 EOF 是一个合法整数, 应该用 ferror 来测试 putw 的错误。

可移植性 所有函数都适用于 UNIX 系统。putc 和 putchar 在 Kernighan 和 Ritchie 书中定义。

参 见 ferror, foper, fread, getc, printf, puts, setbuf

putch 输出字符到控制台

-- putch.c

用 法 int putch(int ch);

原 型 在 conio.h

说 明 见 putc

源 程 序

```
#include <conio.h>
#ifdef _OLDCONIO_
#pragma inline
#include <dos.h>
int
putch(int c)
{
    _DL = (unsigned char)c;
    _AH = 2; /* putch */
    _geninterrupt(0x21);
    return c;
}
```

```

#else
#include < video.h>
int putch(int c)
{
    return __cputn((char*)&c,1,0);
}
#endif

```

putchar 输出字符到流中

用 法 int putch(int ch);

原 型 在 conio.h

说 明 见 putc

putenv 把字符串加到当前环境中

—putenv.cas

用 法 int putenv(char *envvar);

原 型 在 stdlib.h

说 明 见 getenv

源 程 序

```

#pragma inline
#include < asmrules.h>
#include < stdlib.h>
extern char **environ;
extern unsigned _envSize;
int putenv(const char *nameP)
{
    char **envP;
    /* Compute nameP length and remember first character */
asm    LES    di, nameP
#if (LDATA)
asm    mov    ax, ES
asm    or     ax, di
#else
    ES = DS;
asm    or     di, di
#endif
asm    jz     BadPutEnvExit
asm    mov    ah, ES [di]
asm    mov    al, '='
asm    mov    cx, -1
asm    cld
asm    repne  scasb
asm    not    cx
asm    dec    cx
asm    jz     BadPutEnvExit /* CX = VarName length */
    /* Look for nameP in environment array */
#if _HUGE
asm    mov    di, seg environ
asm    mov    DS, di
#endif
asm    LES    di, DPTR_(environ)
#if (LDATA)
asm    mov    W1 (envP), ES
asm    mov    bx, ES
asm    or     bx, di
#else
asm    or     di, di
#endif
asm    mov    W0 (envP), di
asm    jnz    FirstVariable
    /* If no match can be found, return -1. */
BadPutEnvExit:
asm    mov    ax, -1
asm    jmp    PutEnvExit
    /* Does the first character match ??? */

```



```

NextVariable:
asm    add    W0 (envP), dPtrSize
asm    LES    di, envP
FirstVariable:
asm    LES    di, ES_ [di]
#if (LDATA)
asm    mov    bx, ES
asm    or     bx, di
#else
asm    or     di, di
#endif
asm    jz     VarNotFound
asm    mov    al, ES_ [di]
asm    or     al, al    /* "" terminates environment */
asm    jz     VarNotFound
asm    cmp    ah, al    /* compare first characters */
asm    jne    NextVariable
asm    mov    bx, cx
asm    cmp    BY0 (ES_ [di+bx]), '='
asm    jne    NextVariable /* must be followed by '=' */
/* Yes, so compare the remainder of nameP */
asm    pushDS
asm    LDS    si, nameP
asm    repe    cmpsb
asm    popDS
asm    xchg    cx, bx
asm    jne    NextVariable
/* The entries match names. Replace the old pointer with the new. */
VarFound:
asm    LES    di, envP
asm    mov    ax, W0 (nameP)
asm    mov    W0 (ES_ [di]), ax
#if (LDATA)
asm    mov    ax, W1 (nameP)
asm    mov    W1 (ES_ [di]), ax
#endif
asm    jmp     short    GoodPutEnvExit
/* If arrived here then no matching name exists in the table.    A new */
/* entry must be made. Is the existing table big enough ?    */
VarNotFound:
asm    mov    bx, W0 (envP)
asm    sub    bx, W0 (environ)
asm    add    bx, dPtrSize
asm    cmp    bx, _envSize
asm    jb     Append
/* The environment table is full.    It must be enlarged by copying to */
/* a new memory block.    */
asm    add    bx, 4 * dPtrSize
asm    push    bx
asm    call    EXTPROC (malloc)
asm    pop     cx
asm    mov    di, ax
#if (LDATA)
asm    mov    ES, dx
asm    mov    bx, ax
asm    or     bx, dx
#else
asm    push    DS
asm    pop     ES
asm    or     di, di
#endif
asm    jz     BadPutEnvExit
asm    xchg    cx, _envSize
asm    push    cx
asm    pushDS
asm    LDS    si, DPTR_ (environ)
asm    shr     cx, 1
asm    rep     movsw
asm    popDS

```

```

asm    xchg    ax, W0 (environ)
#if (LD=1)
asm    xchg    dx, W1 (environ)
asm    push    dx
#endif
asm    push    ax
asm    call    EXTPROC(free)
asm    add     sp, dPtrSize
asm    pop     bx
/* If arrived here then BX points to the first unused table slot */
/* and the new entry is to be appended. The append is done */
/* by copying the final pointer (to the empty string) up */
/* one place and inserting the new string in its place. */
Append:
asm    LES     di, DPTR (environ)
asm    mov     ax, W0 (nameP)
asm    xchg    ax, W0 (ES - [di+bx-dPtrSize])
asm    mov     W0 (ES - [di+bx]), ax
#if (LDATA)
asm    mov     ax, W1 (nameP)
asm    xchg    ax, W1 (ES - [di+bx-dPtrSize])
asm    mov     W1 (ES - [di+bx]), ax
#endif
GoodPutEnvExit:
asm    xor     ax, ax
PutEnvExit:
    return(_AX);
}

```

putimage 在屏幕上显示一个位图像

用 法 `#include <graphics.h>`
`void far putimage(int x,int y,void far *bitmap,`
`int op);`

原 型 在 `graphics.h`

说 明 见 `getimage`

putpixel 在指定坐标处画一像素

用 法 `#include <graphics.h>`
`void far putpixel(int x,int y,int pixelcolor);`

原 型 在 `graphics.h`

说 明 见 `getpixel`

puts 送一字符串到流中 -- puts.c

用 法 `int puts(char *string);`

相关函数 `void cputs(char *string);`

用 法 `int fputs (char *string,FILE *stream);`

原 型 在 `stdio.h` (`fput,puts`)
`conio.h` (`cputs`)

说 明 `puts` 拷贝以空字符终结的字符串 `string` 到标准输出流 `stdout` 中, 并加上换行符。

`cputs` 写以空字符终结的字符串 `string` 到控制台中, 它不加上换行符。

`fputs` 拷贝以空字符终结的字符串 `string` 到指定名输出流 `stream` 中, 它不加上换行符。

返 回 值 执行成功时, `puts` 和 `fputs` 返回最后写字符; 否则返回 `EOF`。

`cputs` 不返回值

可移植性 这些函数适用于 UNIX 系统, 在 kernighan 和 Ritchie 书中也定义了 `fputs`。

参 见 ferror, fopen, fread, gets, open, printf, putc

源 程 序

```
#include <stdio.h>
#include <string.h>
#include <_stdio.h>
int puts (const register char *s)
{
    if (_fputn (s, strlen (s), stdout))
        return EOF;
    return ((fputc ('\n', stdout) != '\n') ? EOF : '\n');
}
```

puttext 将文本从存储区拷贝到屏幕 -- gptext.c

用 法 int puttext(int left,int top,int right,int bottom,
void *source);

原 型 在 conio.h

说 明 见 gettext

```
源 程 序#include <_video.h>
#include <conio.h>
int puttext(int left, int top, int right, int bottom, void *buffer)
{
    int y, size;
    size = right-left+1;
    for (y = top; y <= bottom; y++)
    {
        screenio( _vptr(left, y), buffer, size);
        (char *)buffer += size*2;
    }
    return 1;
}
```

putw 把一字符或字送到流中 -- putw.c

用 法 #include <stdio.h>
int putw(int w,FILE *stream);

原 型 在 stdio.h

说 明 见 putc

源 程 序

```
#include <stdio.h>
int putw(int w, FILE *fp)
{
    if (putc(((unsigned char *)&(w)), fp) != EOF)
        if (putc(((unsigned char *)&(w) + 1), fp) != EOF)
            return(w);
    return(EOF);
}
```

qsort 使用快速排序例程进行排序 -- qsort.cas

用 法 void qsort(void *base,int nelem,int width,
int (*fcmp)());

原 型 在 stdlib.h

说 明 qsort 实现了“中树遍历”快速排序算法。它通过重复调用用户定义的比较函数（由 fcmp 所指），对表中的元素进行排序。

- * base 指向待排序表的起始位置（第 0 个元素）
- * nelem 是表中的元素个数
- * width 是以字节为单位的表中每个元素长度

*fcmp 比较函数接受两个参数 elem1 和 elem2，均为指向表中元素的指针。比较

函数比较这两指针的内容(*elem1 和 *elem2), 根据比较结果, 返回一整数。

如果	fcmp返回
*elem1 < *elem2	整数 < 0
*elem1 == *elem2	0
*elem1 > *elem2	整数

在比较过程中, 小于(<)符号表示在最后排完序的序列中, 左边元素在右边元素之前出现; 同样, 大于符号(>)表示在最后排完序的序列中, 左边元素出现在右边元素之后。

返回值 qsort 不返回值

可移植性 适用于 UNIX 系统

参 见 bsearch lsearch

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <stdlib.h>
typedef int cdecl comparF (const void *, const void *);
static comparF *Fcmp;
static unsigned qWidth;
void qsort (void *baseP, size_t nElem, size_t width, comparF *compar)
{
    if ((qWidth == width) == 0) return;
    Fcmp = compar;
    qSortHelp (baseP, nElem);
}
```

函数名 qSortHelp - 执行快速排序。 -- qsort.cas

用 法 static void near pascal qSortHelp (char *pivotP,
size_t nElem);

说 明 在 pivotP 所指的 nElem 元素数组上执行快速排序。

返回值 无。

源 程 序

```
static void near pascal qSortHelp (char *pivotP, size_t nElem)
{
    #if defined(_HUGE)
        char *huge *leftP, huge *rightP;
    #else
        char *leftP, *rightP;
    #endif
    unsigned lNum;
    #pragma warn -sus /* avoid warnings in huge model */
    #pragma warn -sig
    tailRecursion:
        if (nElem < 2)
        {
            if (nElem == 2)
            {
                if (Fcmp (pivotP, rightP - qWidth + pivotP) > 0)
                {
                    Exchange (pivotP, rightP);
                }
            }
            return;
        }
        rightP = (nElem - 1) * qWidth + pivotP;
        leftP = (nElem >> 1) * qWidth + pivotP;
        /* sort the pivot, left, and right elements for "median of 3" */
        if (Fcmp (leftP, rightP) > 0)
            Exchange (leftP, rightP);
        if (Fcmp (leftP, pivotP) > 0)
            Exchange (leftP, pivotP);
        else if (Fcmp (pivotP, rightP) > 0)
            Exchange (pivotP, rightP);
        if (nElem == 3)
            return;
        lNum = 1;
        while (leftP < rightP)
        {
            while (Fcmp (leftP, pivotP) < 0)
                leftP = (leftP + 1) * qWidth + pivotP;
            while (Fcmp (rightP, pivotP) < 0)
                rightP = (rightP - 1) * qWidth + pivotP;
            Exchange (leftP, rightP);
            leftP = (leftP + 1) * qWidth + pivotP;
            rightP = (rightP - 1) * qWidth + pivotP;
            lNum++;
        }
        Exchange (pivotP, rightP);
        if (lNum > 1)
            qSortHelp (leftP, lNum * qWidth + pivotP);
        qSortHelp (rightP, (nElem - lNum) * qWidth + pivotP);
}
```

```

        {      Exchange (pivotP, leftP);
                return;
        }
/* now for the classic Hoare algorithm */
leftP = qWidth + pivotP;
do
{
    while (Fcmp (leftP, pivotP) < 0)
        if (_LT_ (leftP, rightP))
            leftP += qWidth;
        else
            goto qBreak;
    while (_LT_ (leftP, rightP))
        if (Fcmp (pivotP, rightP) <= 0)
            rightP -= qWidth;
        else
        {
            Exchange (leftP, rightP);
            leftP += qWidth;
            rightP -= qWidth;
            break;
        }
} while (_LT_ (leftP, rightP));
qBreak:
if (Fcmp (leftP, pivotP) < 0)
    Exchange (leftP, pivotP);
INum = (leftP - pivotP) / qWidth;
if (nElem != INum)
    qSortHelp (leftP, nElem);
nElem = INum;
goto tailRecursion;
#pragma warn .sus
#pragma warn .sig
}

```

raise 向正在执行的程序发送一个信号 -- signal.c

用法 int raise(int sig)

原型在 signal.h

说明 raise 函数允许程序在执行过程中给自己发一个信号，以便处理机执行该信号所规定的动作（由函数 signal 设定），以下是已定义的信号类型：

SIGABRT(22)	退出
SIGFPE(8)	浮点陷阱
SIGILL(4)	非法指令
SIGINT(2)	中断
SIGSEGV(11)	存储存取违反
SIGTERM(15)	终止

用户可以自己设置信号，详见 signal 示例

参 见 signal

源 程 序

```

int raise(int SigType)
{
    int      Index;
    CatcherPTR action;
    if ((Index = GetIndex(SigType)) == BogusSignal)
        return 1;
    if ((action = Dispatch[Index]) != SIG_IGN)
        if (action == SIG_DFL)
            switch (SigType)
            {
                case SIGABRT :
                    _exit(3);
#ifdef __OS2__
                case SIGBREAK :

```

```

#endif
                case SIGINT      :
#ifdef __OS2__
                /* OS/2 has no analog to int 23H */
                _exit(1);
#else
                geninterrupt( 0x23 );
#endif
                break;
#ifdef __OS2__
                case SIGUSR1      :
                case SIGUSR2      :
                case SIGUSR3      :
                break;
#endif
                case SIGILL       :
                case SIGSEGV      :
                case SIGTERM      :
                case SIGFPE       :
                default            :
                _exit(1);
            }
        else
        {
            /* Call user routine. Add optional parameter
            specifying that the signal was raised explicitly
            rather than asynchronously. */
#ifdef ANSI_CONFORMING
            Dispatch[Index] = SIG_DFL; /* Always default (ANSI) */
#else
            if (SigType != SIGFPE) /* Maybe default (MS) */
                Dispatch[Index] = SIG_DFL;
#endif
        }
        (*action)(SigType, ExplicitVal[Index]);
    }
    return 0;
}

```

rand 随机数发生器 -- rand.c

用 法 int rand(void);

相关函数

用 法 void srand(unsigned seed);

原 型 在 stdlib.h

说 明 rand 使随机数发生器产生一范围从 0 到 2 的伪随机数。

发生器通过调用参数值为 1 的 srand 函数可重初始化, 也可以用给定的起始值。

seed 调用 srand 产生一新的起点。

可移植性 适用于 UNIX 系统

源 程 序

```

#include <stdlib.h>
#define MULTIPLIER    0x015a4e35L
#define INCREMENT     1
static long Seed = 1;
int rand(void)
{
    Seed = MULTIPLIER * Seed + INCREMENT;
    return((int)(Seed >> 16) & 0x7fff);
}

```

randbrd 随机块读 -- randblk.cas

用 法 #include <dos.h>

int randbrd(struct fcb *fcbptr, int recnt);

相关函数

用 法 `int randbwr(struct fcb *fcbptr, int recent);`

原 型 在 `dos.h`

说 明 `randbwr` 使用 `fcbptr` 所指的打开的 FCB 读 `recent` 个记录。记录被读到当前磁盘传输地址所在的存储区。它们是从 FCB 的随机记录(`random record`)字段指明的磁盘记录中读到的, 由 DOS 系统调用 `0x27` 完成。

`randbwr` 与 `randbrd` 的功能相似, 只是它是写数据到磁盘中间不是从磁盘读数据。由 DOS 系统调用 `0x28` 完成。如果 `recent` 为 0, 则文件被截成所指明的大小。

实际读写的记录可通过检查 FCB 的随机记录域而确定, 随机记录域值随着实际读写的记录数而变。

返 回 值 根据 `randbrd` 或 `randbwr` 操作的结果返回下列值:

- 0 读或写了所有记录;
- 1 到达文件末尾, 最后一个记录读完成;
- 2 读记录到 `0xFFFF` 地址后反绕 (读尽可能多的记录);
- 3 跳过文件末尾, 最后一个记录不完整。

当没有足够的磁盘空间用于写记录时, `randbwr` 返回 1 (不写记录)

可移植性 只适用于 MS-DOS

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
int randbrd(struct fcb *fcb, int recent)
{
    pushDS
    asm    mov     ah, 027h
    asm    mov     cx, recent
    asm    LDS     dx, fcb
    asm    int     021h
    popDS
    return (_AX & 0xFF);
}
```

`randbwr` 随机块写

-- `randblk.cas`

用 法 `#include <dos.h>`

`int randbwr(struct fcb *fcbptr, int recent);`

原 型 在 `dos.h`

说 明 见 `randbrd`

源 程 序

```
int randbwr(struct fcb *fcb, int recent)
{
    pushDS
    asm    mov     ah, 028h
    asm    mov     cx, recent
    asm    LDS     dx, fcb
    asm    int     021h
    popDS
    return (_AX & 0xFF);
}
```

`random` 随机数发生器

用 法 `#include <stdlib.h>`

`int random(int num);`

相关函数

用 法 void randomize(void);

原 型 在 stdlib.h

说 明 random 返回一个从 0 到(num-1)的随机数。random(num)是一个定义为 rand()%(num)的宏。num 和返回的随机数均为整数。

randomize 用一个随机值对随机数发生器进行初始化。由于 randomize 是作为宏来实现的，该宏调用了原型在 TIME.H 中的 time 函数，因此在使用本函数时，应使用 #include <time.h>

返 回 值 random 返回一个从 0 到(num-1)之间的数，randomize 不返回任何值

可移植性 在 Turbo Pascal 中有相似的程序

参 见 rand,seed

randomize 对随机函数发生器进行初始化

用 法 #include <stdlib.h>

void randomize(void);

原 型 在 stdlib.h

说 明 见 random

_read 从文件中读

-- reada.cas

用 法 int _read(int handle,void *buf,int nbyte);

原 型 在 io.h

说 明 见 read

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
int _read (int fd, void *buf, unsigned len)
{
    pushDS_
    asm    mov     ah,3fh
    asm    mov     bx,fd
    asm    mov     cx,len
    asm    LDS     dx, buf
    asm    int     21h
    popDS_
    asm    jc      readFailed
    return(_AX);
_readFailed:
    return __IOerror (_AX);
}
```

read 从文件中读

-- read.cas

用 法 int read (int handle,void *buf,int nbyte);

相关函数

用 法 int _read(int handle,void *buf,int nbyte);

原 型 在 io.h

说 明 read 和 _read 从与 handle 相联的文件中读入 nbyte 字节到由 buf 所指的缓冲区中，可读的最大字节数为 65534，因为 65535(0xFFFF)与这两个函数的错误返回标志 -1 相同。

_read 直接调用 MS-DOS 读系统调用。

对于用文本方式打开的文件，read 删除回车符，当读到 Ctrl_Z 字符时报告文件结束。_read 不执行这些功能。

handle 是从 creat, open, dup, dup2 或 fcntl 调用中得到的句柄。

对于磁盘文件, 这些函数从当前文件指针开始读。在读完成后, 文件指针增量为读入的字节数。

对于设备, 字节直接从设备中读。

返回值 执行成功时, 返回一正整数表示读入缓冲区的字节数。如果文件以文本方式打开, 所读字节数不包括回车和 Ctrl-z 字符。

在文件末尾时, 两个函数均返回 0。

在出错时, 两个函数均返回 -1。且置 errno 为下列值之一:

EACCES	无此权限
EBADF	无效文件号

可移植性 read 适用于 UNIX 系统。

_read 只适用于 MS-DOS。

参见 creat, dup, fread, getc, open

源程序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <fcntl.h>
#define _ctlZ 26
#pragma warn -use
int read (int fd, void *buf, unsigned int len)
{
    #if 0
        char    *top;
        char    *bottom;
    #endif
        register unsigned    dlen;
        char    c;
        register DI;          /* prevent compiler usage of registers */
        if ((len + 1) < 2 || !_openfd[fd] & O_EOF)
            return 0; /* 0, -1 are not allowed lengths */
        do {
            dlen = _read (fd, buf, len);
            if ((dlen + 1) < 2 || !_openfd[fd] & O_BINARY)
                return (dlen);
            /* Squeeze out carriage returns */
        #if 0
            top = bottom = buf;
            do {
                if ((c = *top++) == _ctlZ) {
                    lseek (fd, -dlen, SEEK_END);
                    _openfd[fd] |= O_EOF; /* note that ^Z EOF has been seen */
                    return bottom - buf;
                }
                else { if (c != '\r')
                        *bottom++ = c;
                    else if (1 == dlen) {
                        read (fd, &c, 1);
                        *bottom++ = c;
                    }
                }
            } while (--dlen);
        } while ((bottom - buf) == 0);
        return bottom - buf;
    #endif
    asm    mov    cx, dlen
    asm    LES    si, buf
    #if (! LDAT)
    asm    push    DS
    asm    pop     ES
    #endif
}
```

```

asm    mov    di, si
asm    mov    bx, si
asm    cld
squeeze:
asm                lods    BYTE [ES: [si]]
asm                cmp     al, ctIZ
asm                je      endSeen
asm                cmp     al, 0Dh          /* '\r' */
asm                je      elseSqueeze
asm                stosb
whileSqueeze:
asm                loop     squeeze
asm                jmp     short squeezeBreak
elseSqueeze:
asm                loop     squeeze          /* if (1 == dlen) */
asm                push     ES
asm                push     bx
asm                mov     ax, 1
asm                push     ax
asm                lea     ax, c
#if LDATA
asm                push     SS
#endif
asm                push     ax
asm                push     fd              /* _read (fd, &c, 1); */
asm                call     EXTPROC (_read)
asm                add     sp, 4 + dPtrSize
asm                pop     bx
asm                pop     ES
asm                cld
asm                mov     al, c
asm                stosb                    /* *bottom++ = c; */
squeezeBreak: ;
    ) while (_DI == _BX);
asm    jmp     short doneText
endSeen:
asm    push    bx                          /* keep safe */
asm    mov     ax, SEEK_END
asm    push    ax
asm    neg     cx
asm    sbb     ax, ax
asm    push    ax
asm    push    cx
asm    push    W0 (fd)
asm    call    EXTPROC(lseek) /* lseek (fd, -dlen, SEEK_END); */
asm    add     sp, 8
asm    _openfd[fd] |= _O_EOF; /* note that ^Z EOF has been seen */
asm    pop     bx
doneText:
    return _DI - _BX;
}
#pragma warn .use

```

函数名 `__realcvt` - 将 double 值转换为 ASCII 字符串。 -- `realcvt.cas`

用法 `void pascal __realcvt (double *valueP, int ndec,
char *strP, char formCh,
char altFormat);`

原型文件 `_printf.h`

说明 double 值通过 F 格式:

[sign][zeroes] ...

或 E 格式:

[sign]digit ...

转换为 ASCII 字符串。

这里前导符号仅为负数、小数点省略($ndec = 0$)或数 $\leq \text{MIN}(1, ndec)$ 时才被插入。

如果 $\text{formCh} == 'f'$ 或 $'g'$ 或 $'G'$ 且有多于三个的前导 0, 并且没有尾随的 0 时使用 F 格式, 否则使用 E 格式。

如果 $\text{formCh} == 'E'$ 或 $'G'$, 则字符'E' 用于从指数中分离小数部分。如果 altForm 为真 (非 0) 则保留尾随的 0, 并且 G/g 格式不删除 0。

如果 $ndec > 18$ 则同 $ndec = 18$ 。

如果值为无穷在则字符串为 "9E+999" 或 "-9E+999"。

对于数字串的基本转换是通过 `__xcvt()` 完成的。因为该程序响应 `__xcvt()` 返回的专用简单格式。

返回值 无。

源程序

```
#pragma warn -use
static void pascal __realcvt (void *valueP, int ndec,
                             char *strP, char formCh, char altFormat, int type)
{
    char    buf [ __XCVTDIG__ + 4];
    int     sign, realcnt;
    register SI, DI;      /* prevent the compiler making its own usage */
    I       push     ES
#ifdef LDATA
    I       push     DS
    I       pop      ES
#endif
    I       mov      ax, ndec
    I       cmp      ax, __XCVTDIG__
    I       jna      meaningful
    I       mov      ax, __XCVTDIG__
    I       mov      ndec, ax
meaningful:
    I       mov      realcnt, ax
    I       mov      dl, formCh
    I       and      dl, 0DFh
    I       cmp      dl, 'F'
    I       jne      overallDigits
    /* F-format works with digits right of the decimal point, specified */
    /* to __xcvt() as a negative number. */
    I       neg      ax
    I       jng      bounded
    I       sub      ax, ax
    I       mov      ndec, ax
    I       jmp      short bounded
    /* E-format counts overall digits, as does g-format sometimes.      There */
    /* must be at least one. */
overallDigits:
    I       or       ax, ax
    I       jg       bounded
    I       mov      ax, 1
    I       jmp      short decimalsDecided /* that one is left of '.' */
bounded:
    I       cmp      dl, 'F'
    I       je       decimalsDecided
    I       inc      ax
    I       inc      W0 (ndec)
    /* eg fmts have digit left of '.' */
    /* so ask __xcvt() for one more. */
decimalsDecided:
    /* __xcvt will do the difficult part, conversion to decimal, but the */
    /* format it returns must be worked on before it can be passed */
    /* to the caller. Therefore put it into a temporary buffer. */
#ifdef LDATA
    I       push     W1 (valueP)
#endif
    I       push     W0 (valueP) /* (valueP */
    I       cmp      ax, 0
    I       jle      ok_cnt
    asm      mov     ax, realcnt
```

```

ok_cnt:
I      push    ax                /*      , realcnt      */
#ifdef LDATA
I      push    SS
#endif
I      lea     bx, sign
I      push    bx                /*      , &sign      */
#ifdef LDATA
I      push    SS
#endif
I      lea     si, buf
I      push    si                /*      , buf        */
I      mov     ax, W0 (type)     /*      , type      */
I      push    ax
I      call    EXTPROC (__xcvt) /* returns exponent in AX */
I      xchg    bx, ax           /* BX = exponent */
/* Set up pointer to destination string */
I      LES     di, strP
I      cld                     /* forward string direction */
/* Check for Infinity and NAN numbers right up front */
I      cmp     bx, INF_number
I      je      infinities
I      cmp     bx, NAN_number
I      je      NANs
I      jmp     short regular_number
/*+-----+
| We have a NAN or Infinity here. Do +INF/-INF or +NAN/-NAN |
+-----+*/
infinities:
I      mov     ax, 492BH         /* 'I+' stosw reverses */
I      cmp     W0 (sign), 0
I      je      store_Isign
I      inc     ax                /* Change 'I+' to 'I-' */
I      inc     ax
store_Isign:
I      stosw
I      mov     ax, 464EH         /* 'FN' stosw reverses */
I      stosw
I      goto    end;

/* Print a NAN */
NANs:
I      mov     ax, 4E2BH         /* 'N+' stosw reverses */
I      cmp     W0 (sign), 0
I      je      store_Nsign
I      inc     ax                /* Change 'N+' to 'N-' */
I      inc     ax
store_Nsign:
I      stosw
I      mov     al, 'A'           /* AX = 'NA' stosw reverses */
I      stosw
I      goto    end;

/*+-----+
| We're printing a regular number |
+-----+*/
regular_number:
/* Either format begins with optional sign. */
I      cmp     BY0 (sign), 0
I      je      signSet
I      mov     al, '-'
I      stosb
signSet:
/* Now that we have the basic string, decide what format the caller */
/* wants it to be put into. */
I      mov     dl, formCh        /* possibilities are e, E, f, g, or G */
I      and     dl, 5Fh           /* convert to upper case */
I      cmp     dl, 'F'
I      je      F_format
I      cmp     dl, 'E'
I      je      E_format

```

```

/* G format can be either E or F, depending on circumstances. */
I    cmp    bx, -3          /* Now decide between E and F formats. */
I    jl     E_format
I    cmp    bx, ndec
I    jg     E_format
/* The F format has no written exponent and the decimal point can be at */
/* the left edge or any interior position, but not the right edge. */
F_format:
I    cmp    bx, __XCVDIG__ /* refuse to do F format if more than */
I    jg     E_format      /* __XCVDIG__ integral digits can result. */
I    or     bx, bx
I    jg     FdigitStart
/* No integral digits, begin with '0.'. */
I    mov    ax, ('.' SHL 8) + '0'
I    stosw
I    mov    cx, 1
I    je     Fdigits
/* If the exponent is negative then leading zeroes are required. */
I    mov    al, '0'
FleadZeroes:
I    stosb
I    inc    bx
I    jnz    FleadZeroes
/* Now write the regular digits, inserting a '.' if it is somewhere */
/* in the middle of the numeral. */
FdigitStart:
I    mov    cx, 0
Fdigits:
I    lods   BY0 (SS_ [si])
I    or     al, al
I    jz     Fend
I    stosb
I    dec    bx
I    jnz    Fdigits
I    mov    al, '.'
I    stosb
I    inc    cx
I    jmp    short Fdigits
/* remove any trailing zeroes, unless there is an implied decimal at */
/* the right edge. */
Fend:
I    mov    ax, ndec
I    add    cx, realcnt
I    cmp    ax, cx
I    jbe    no_zero_pad
I    sub    ax, cx
I    mov    cx, ax
I    add    bx, ax
I    mov    al, '0'
I    rep    stosb
I    dec    bx
I    jz     Fz
no_zero_pad:
I    dec    bx
I    jz     Fz
I    cmp    BY0 (altFormat), 0
I    jne    Fz          /* altFormat implies no trimming */
I    mov    di, formCh
I    mov    cx, W0 (strP)
I    xchg   bx, di
TrimTrailing ();          /* backspaces BX to effect trimming */
I    xchg   di, bx
Fz:          /* is the result an empty string ? */
I    cmp    di, W0 (strP)
I    jne    goto_end
I    mov    al, '0'
I    stosb          /* if so, it is a form of zero. */
goto_end:
I    jmp    end      /* the string will be zero terminated */

```

```

/* The E format always places one digit to the left of the decimal
point, followed by fraction digits, and then an 'E' followed
by a decimal exponent. The exponent is always 2 digits unless
it is of magnitude > 99. */
E format:
I lods W0 (SS_ [si]) /* get two bytes together */
I stosb
/* The decimal point appears only if followed by a digit. */
I mov al, '.'
I or ah, ah
I jz Eexp
/* If arrived here then there are at least two digits so put in */
/* the decimal point and copy the fraction digits. */
I stosb
I mov al, ah
Edigits:
I stosb
I lods BY0 (SS_ [si])
I or al, al
I jnz Edigits
/* Trailing zeroes are removed from the fraction. */
I cmp BY0 (altFormat), 0
I jne Eexp /* altFormat implies no trimming */
I mov dl, formCh
I mov cx, W0 (strP)
I xchg bx, di
TrimTrailing (); /* backspaces BX to effect trimming */
I xchg di, bx
/* Now put in the exponent. Note that the exponent returned from __xcvt */
/* is one digit different, since __xcvt places the decimal point at */
/* the left edge. */
Eexp:
I mov al, 20h /* the bit which distinguishes lower case */
I and al, formCh /* e f g E G -- cause */
I or al, 'E' /* e e e E E -- effect */
I stosb
I mov ax, 2D2Bh /* AX = "-+" load up both possibilities */
I dec bx /* BX is the exponent returned from __xcvt */
I jnl EexpSigned
I xchg al, ah /* It was negative, switch signs */
I neg bx /* Make it positive for later */
EexpSigned:
I stosb
I xchg ax, bx
/* ANSI says that "The exponent always contains AT LEAST two digits".
If the exponent is bigger than 99 then we will use as many as are
required. */
I mov cx, 3030h /* "00" this'll be handy later */
I cmp ax, 99
I jna EtwoDigits
I cmp ax, 999
I jna EthreeDigits
EfourDigits:
I cwd /* Sign extend AX through DX */
I mov bx, 1000
I div bx
I add al, cl
I stosb
I xchg ax, dx
EthreeDigits:
I mov bl, 100
I div bl
I add al, cl
I stosb
I xchg al, ah
I cbw
EtwoDigits:
I mov bl, 10
I div bl

```

```

I      add     ax, cx      /* '00' */
I      stosw
/* Both formats are terminated with \0. */
end:
I      xor     al,al
I      stosb
I      pop     ES
      return;
}
#pragma warn .use

```

realloc 重分配存储区

-- realloc.c

用 法 void *realloc (void *ptr,unsigned newsize);

原 型 在 stdlib.h,alloc.h

说 明 见 malloc

源 程 序

```

#if defined(__TINY__) || defined(__SMALL__) || defined(__MEDIUM__)
#include <mem.h>
#include "heap.h"
/* reallocates a block */
void *cdecl realloc( void *q, unsigned int newsize )
{
    unsigned int oldsize;
    struct header *oldq;
    void *newq;
    oldq = (struct header *)
        ((char *)q - USED_HEADER_SIZE); /* calc block's start address */
    oldsize = oldq->size - 1 - USED_HEADER_SIZE;
    newq = malloc( newsize );
    if( newq )
    {
        movmem( q, newq, oldsize < newsize ? oldsize : newsize );
        free( q );
    }
    return( newq );
}
#endif

```

rectangle 画一个矩形

用 法 #include <graphics.h>

void far rectangle(int left,int top ,int right,int bottom);

原 型 在 graphics.h

说 明 rectangle 用当前线型、宽度和画线颜色画一矩形。

矩形的左上角为(left,top), 右下角为(right,bottom)。

返 回 值 无

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 bar,getlinesettings,getcolor

registerbgidriver 注册已连接进来的图形驱动程序代码 -- gregistr.c

用 法 #include <graphics.h>

int registerbgidriver(void (*driver)(void));

相关函数

用 法 int registerbgifont(void (*font)(void));

说 明 registerbgidriver 通知图形系统存在着一个连入的驱动程序。同样, 调用 registerbgifont 表示

有一个连入的矢量字体文件。这些例行程序检查指定驱动程序或字体的连接代码。如果代码有效，就把它登记到内部表中。通过在对 `registerbgidriver` 或者 `registerbgifont` 的调用过程中使用已连接文件的名，用户可以使用该公共名告诉编译程序、连接程序连接目标代码。

返回值 如果指定了无效的驱动程序或字体，本函数返回负的图形错误码；否则，`registerbgifriver` 返回一个内部驱动程序号，`registerbgifont` 返回已注册的字体号。

可移植性 在 Turo Pascal 5.0 中有相似的子程序

参 见 `initgraph`, `gettextsettings`

源 程 序

```
#include <_graph.h>
#include <process.h>
void registerbgidriver(void (*driver)(void))
{
    registerfarbgidriver((void far *)driver);
}
void registerbgifont(void (*font)(void))
{
    registerfarbgifont((void far *)font);
}
```

remove 删掉一个文件

用 法 `int remove(char *filename);`

原型在 `stdio.h`

说 明 见 `unlink`

rename 重命名文件

-- `rename.cas`

用 法 `int rename(char *oldname, char *newname);`

原型在 `stdio.h`

说 明 `rename` 把老文件名 `oldname` 改为新文件名 `newname`。如果 `newname` 中包含了一个驱动器指示符，它必须与 `oldname` 中给出的相同。

路径中的目录可以不同，因此 `rename` 可用于把文件从一个目录移到另一个。

不允许有 DOS 匹配符 (*, ? 等)

返回值 若重命名文件成功，`rename` 返回 0；出错时，返回 -1，并置 `errno` 为下列值之一：

<code>ENOENT</code>	路径或文件名没有找到
<code>EACCES</code>	无此权限
<code>ENOTSAM</code>	不是同一设备

可移植性 只适用于 MS-DOS

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <stdio.h>
#include <io.h>
int rename(const char *oldname, const char *newname)
{
    #if !LDATA
        _ES = _DS;
    #endif
    pushDS
    asm mov ah, 056h
    asm LDS dx, oldname
    asm LES di, newname
    asm int 021H
    popDS
    asm jc renameFailed
```



```

        return(0);
renameFailed:
        return __IOerror(_AX);
}

```

restorecrtmode 将屏幕模式恢复为 initgraph 设置的值。

用 法 #include <graphics.h>
 void far restorecrtmode(void);

原 型 在 graphics.h

说 明 restore 恢复 initgraph 检测到的原视屏模式，它可以与 setgraphmode 一起使用，在文本和图形模式之间进行转换。

返 回 值 无

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 initgraph, setgraphmode

rewind 重定位流 -- rewind.c

用 法 #include <stdio.h>
 int rewind(FILE *stream);

原 型 在 stdio.h

说 明 见 fseek

源 程 序

```

#include <stdio.h>
void rewind (FILE *fp)
{
    if (fseek (fp, 0L, SEEK SET) == 0)
        fp->flags &= ~_F_ERR;
}

```

rmdir 删除目录 -- rmdir.cas

用 法 int rmdir (char *pathname);

原 型 在 dir.h

说 明 见 mkdir

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <io.h>
int rmdir(const char *pathP)
{
    pushDS_
    asm    mov     ah, 03Ah
    asm    LDS     dx, pathP
    asm    int     021H
    popDS_
    asm    jc      rmdirFailed
    return(0);
rmdirFailed:
    return __IOerror(_AX);
}

```

_rotl 将一个值向左循环移位 -- rotl.cas

用 法 unsigned _rotl(unsigned value, int count);

相关函数 unsigned _rotr(unsigned value, int count);

用 法 unsigned long _lrotl(unsigned long lvalue,int count);
unsigned long _lrotr(unsigned long lvalue,int count);

原 型 在 stdlib.h

说 明 这些函数将给定的值 value 向左或向右循环移动 count 位。对于 _lrotl 和 _lrotr, lvalue 为 unsigned long 型; 对于 _rotl 和 _rotr, 其循环移位值为 unsigned 型。

_rotl 将 value 向左循环移动 count 位;
_rotr 将 value 向右循环移动 count 位;
_lrotl 将 lvalue 向左循环移动 count 位;
_lrotr 将 lvalue 向右循环移动 count 位。

返 回 值 这些函数返回移位后的值。

源 程 序

```
#pragma inline
#include <stdlib.h>
unsigned _rotl(unsigned val, int rotate_count)
{
asm      mov     ax, val
asm      mov     cx, rotate_count
asm      rol     ax, cl
      return    _AX;
}
```

_rotr 将一个值向右循环移位 -- rotr.cas

用 法 unsigned _rotr(unsigned value ,int count);

原 型 在 stdlib.h

说 明 见 _rotl

源 程 序

```
#pragma inline
#include <stdlib.h>
unsigned _rotr(unsigned val, int rotate_count)
{
asm      mov     ax, val
asm      mov     cx, rotate_count
asm      ror     ax, cl
      return    _AX;
}
```

__sbrk 在近堆栈中修改数据段空间申请。 -- brk.cas

用 法 void *__sbrk(long incr);

原型文件 alloc.h

说 明 将终止值增加 incr 字节并相应修改已申请的空间。
incr 可以为负, 这时已申请的空间减小。

返 回 值 成功: 旧终止值;

失败: -1, 并将 errno 设置为 ENOMEM(无足够空间)。

源 程 序

```
void *__sbrk(long incr) {
asm      mov     ax, W0(incr)
asm      mov     dx, W1(incr)
asm      add     ax, word ptr __brklvl
asm      adc     dx, 0
asm      mov     cx, ax
asm      add     cx, MARGIN
asm      adc     dx, 0
asm      or      dx, dx
```

```

asm    jnz     sbrkErr
asm    cmp     cx, sp
asm    jnb     sbrkErr
asm    xchg    word ptr _brkval, ax
asm    return (void *)_AX;
sbrkErr:
    errno = ENOMEM;
    return((void *)-1);
}

```

_sbrk 修改数据段空间申请。 -- fbrk.c

用 法 void huge *_sbrk(long incr);

原型文件 alloc.h

说 明 将终止值增加 incr 个字节并相应修改已申请空间。incr 可以为负，这时申请的空间量减少。

返 回 值 成功：旧的终止值；

失败：-1，并将 error 设置为 ENOMEM(空间不够)。

源 程 序

```

void    huge *_sbrk(long incr)
{
    char    huge    *cp;
    char    huge    *old;
    cp = _brkval + incr;
    if ((cp < heapbase) || (cp > heaptop))
        return((void huge *)-1L);
    old = _brkval;
    if (!normalize((char far *)cp))
        return((void huge *)-1L);
    return(old);
}

```

sbrk 改变数据段空间位置 -- brk.cas

用 法 char *sbrk(int incr); 原型在 alloc.h

说 明 见 brk

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <alloc.h>
#include <errno.h>
void    *sbrk(int incr)
{
    return(__sbrk((long)incr));
}
#endif

```

...scanf 执行格式化输入 -- scanf.c

用 法 int scanf(char *format[,argument,...]);

相关函数 int cscanf(char *format[,argument,...]);

用 法 int fscanf(FILE *stream,char *format [,argument,...]);

int sscanf(char *string,char *format[,argument,...]);

int vfscanf(FILE *stream,char *format,va_list argp);

int vscanf(char *format,va_list argp);

int vsscanf(char *string,char *format,va_list argp);

原 型 在 stdio.h

说 明 ...scanf 系列函数扫描输入字段，一次一个字符，再按一定格式转换。这些功能

有：

- 接受一格式字符串确定输入字段的解释方式（由用法中的 format 给定）
- 为了格式输入，对数量可变的输入字段提供格式字符串
- 将格式后的输入存储在格式字符串后的参数给出的地址（这些地址在用法中以“argument”或 va_list param 形式给出）

当...scanf 函数在格式字符串中遇到它的第一个格式指示符时，它根据该指示符扫描并转换第一个输入字段，然后把结果存在由第一个地址参数给定的位置中；接着转换并存储第二个输入字段，第三个...等等。

...scanf 中三个函数的输入流是隐含的。

scanf 和 vscanf 都从 stdin 接受输入。

cscanf 直接从控制台接受输入。

另四个...scanf 函数也使用另一个参数（参数表中的第一个参数），这个附加参数指定输入源：

fscanf 和 vfscanf 从流（由 stream 所指）中接受输入。

sscanf 和 vsscanf 从内存字符串（由 stream 所指）中接受输入。

...scanf 的四个函数直接从函数调用中设置地址（scanf, cscanf, fscanf, sscanf）

其余三个函数（vscanf, vfscanf, vsscanf）从可变参数表中得到地址参数。v...scanf 函数为...scanf 函数的可选择入口点。

有关参数表的更详细信息，见 va... 的定义。

以下对每一...scanf 函数作一总结：

- scanf 从 stdin 读数据，然后把它存在地址参数 &arg1, ..., &argn 给定的位置中
- cscanf 直接从控制台读数据，然后把它存在地址参数 &arg1, ..., *argn 给定的地址中
- fscanf 从指定名输入流中读数据，然后把它存在地址参数 &arg1, ..., &argn 给定的地址中
- sscanf 读数据（存在字符串 string 中）到地址参数 &arg1, ..., &argn 给定的位置，不改变源串
- vscanf 同 scanf 相似，只是从 va_arg 数组 va_list param 中接受地址参数
- vfscanf 同 fscanf 相似，只是它从 va_arg 数组中接受地址参数
- vsscanf 同 sscanf 相似，只是从 va_arg 数组 va_list param 中接受地址

1. 格式字符串；

在每一...scanf 函数调用中出现的格式字符串用于控制函数扫描、转换和存储输入字段的方式。对于给定的格式指示符，必须有足够的地址参数；否则的话，结果是不可预测的，也可能是灾难性的。多余的地址参数（多于格式所要求的）被忽略。

格式字符串是包含三种类型目标的字符串：空白字符，非空白字符和格式指示符。

- 空白字符为空格()、制表(\t)或换行符(\n)。如果一个...scanf 函数在格式字符串中遇到一个空白字符，它将读但不存所有连续的空白字符直到输入中出现非空白字符。
- 非空白字符是除了百分号(%)外的所有其它 ASCII 字符。如果一个...scanf 函数在格式字符串中遇到一个非空白字符，它将读但不存匹配的非空白字符。
- 格式指示符控制...scanf 函数读入并转换输入字段中的字符为给定类型的值，然后把它们存在由地址参数给出的位置上。

末尾的空格（包括换行符）不读，除非在格式字符串中显式匹配。

2. 格式指示：

...scanf 格式指示符有以下形式：

%[*] [width] [F/N] [h/l] type_character

每一格式指示以百分号(%)开始，接着是如下顺序：

- 一个可选的赋值抑制字符 [*]
- 一个可选的宽度指示符 [width]
- 一个可选的指针大小指示符 [F/N]
- 一个可选的参数类型修饰符 [h/l]
- 类型字符

3. 可选的格式字符串成份:

以下是由...scanf 格式字符串中可选的字符和指示符所控制的输入格式的一般形式:

字符或指示符	控制或指示
•	控制下一个输入字段的赋值。
width	可读入的最多字符数。如果...scanf函数遇到了空白字符或不可转换字符, 读入字符将减少。
size	覆盖地址参数的缺省大小 (N=近指针, F=远指针)。
类型参数	覆盖地址参数的缺省类型(h=指向短整型的指针,l=指向长整型的指针)。

4. ...scanf 类型字符:

下表列出...scanf 类型字符每个要求的输入类型和输入的存储格式。

本表假定在格式指示符中没有包括可选的字符、指示符或修饰符(*,width,size)。如果想知道可选择成份是怎样影响...scanf 输入的, 请看接下来的那张表。

类型字符	输入	参数类型
数值		
d	十进制数	指向整型的指针(int *arg)
D	十进制数	指向长整型的指针(long *arg)
o	八进制数	指向整型的指针(int *arg)
O	八进制数	指向长整型的指针(long *arg)
i	十、八或十六进制数	指向整型的指针(int *arg)
I	十、八或十六进制数	指向长整型的指针(long *arg)
u	无符号十进制数	指向无符号整型的指针(unsigned int *arg)
U	无符号十进制数	指向无符号长整型的指针(unsigned long *arg)
x	十六进制数	指向整型的指针(int *arg)
X	十六进制数	指向长整型的指针(long *arg)
e	浮点数	指向浮点型的指针(float *arg)
E	浮点数	指向双精度型的指针(double *arg)
f	浮点数	指向浮点型的指针(float *arg)
g	浮点数	指向浮点型的指针(float *arg)
G	浮点数	指向双精度型的指针(double *arg)
字符		
s	字符串	指向字符数组的指针
c	字符	指向单个字符的指针, 如果同时给定字段宽

(如%Sc), 将为指向w维字符数组的指针
% 字符 不作转换, 存字符%

		指针
n	无	指向整型的指针(int *arg), 成功读入的字符(到%n为止)数被存到此指针中
p	以YYYY:ZZZZ 或ZZZZ形式 表示的十六进制数	指向对象的指针 (far* 或 near*) 转换缺省为存储模式的指针大小

5. 输入字段:

下列之任何一个为输入字段:

- 下一空白字符前的所有字符(不包括空白字符)
- 在当前格式指示下不能转换的第一个字符前的所有字符(如八进制格式下的8或9)
- 到n个字符, 其中n为字段指示宽度

6. 约定

格式指示符中有些约定, 总结如下:

%c 转换:

这个指示符读下一个字符(包括空白字符), 为了跳过空白字符, 读下一个非空白字符, 可使用%ls.

%wc 转换(w=宽度指示):

地址参数是一字符数组指针, 该数组包含w个元素(char arg[w])

%s 转换:

地址参数是一指向字符数组的指针(char arg[])

数组大小至少为n+1个字节, 其中n为字符串s的长度. 用空格或换行结束输入字段, 在数组的最后一个元素后自动加上一空字符终字符.

%[search_set] 转换:

由方括号括起来的字符集合可用s类型字符替代. 地址参数是一个指向字符数组的指针(char arg[]).

方括号内包含了一个可能字符的搜索集合组成的字符串(输入字段).

如果括号内的第一个字符^, 则搜索集合为除去括号内字符的所有其它ASCII字符.

输入字段是不由空格分界的字符串. ...scanf 函数读相应的输入字符, 直到遇上不在搜索集合(或相反搜索集合)中的第一个字符为止. 以下是这种转换类型的两个例子:

%[abcd] 搜索输入字段中所有的a, b, c, d字符

%[^abcd] 搜索输入字段中除a, b, c, d外的其它字符

在搜索集合中, 可以使用“范围设置”来定义一个字符(数字或字母)的范围. 如为搜索所有的数字, 可以使用:

%[0123456789]

也可以使用:

%[0-9]

搜索字母或数字, 可使用:

%[A-Z] 搜索所有大写字母;

%[0-9A-Za-z] 搜索所有数字, 字母(大小写);

%[A-FT-Z] 搜索从A到F, T到Z的大写字母;

控制这些搜索集合范围的规则是很直观的:

- 短横线(-)前的字符必须小于后字符;
- 短横线不能是最前或最后字符(否则就被认为是字符短横线,而不是范围指示符);
- 短横线两边字符必定是范围的起始和终止值;

以下列出短横线仅是字符或是范围指示符的一些例子:

%[-+*/] 四个算术操作符
 %[z-a] 字符'z','-', 'a';
 %[+0-9-A-F] 字符'+','-',范围0到9,A到Z;
 %[-0-9A-F] 同上;
 %[^-0-9+A-F] 除了'+','-',范围0到9,A到Z外的所有字符。

%e,%E,%f,%g,%G(浮点转换)

输入字段中的浮点数具有下列一般格式:

[+/-]ddddddddd[.]ddd[E|e][+|-]ddd

其中[]内为任选项,ddd 代表十进制,八进制或十六进制数字。

%d,%i,%o,%x,%D,%l,%O,%X,%c,%n 转换

在允许使用指向字符、整型或长整型的转换中,都可使用指向无符号字符、无符号整型或无符号长整型的指针。

7 赋值抑制字符:

赋值抑制字符为星号(*),不要把它与 C 的间接操作符(指针)相混淆(也为星号)。

如果格式指示中有*号跟在%号后,下一输入字段被扫描但不赋给下一地址参数。被抑制的输入数据类型假定为跟在*号的类型字符所指定的类型。

后继的文字匹配和赋值抑制不能直接决定。

8 宽度指示符:

宽度指示符(n)为一个十进制整数,它控制从当前输入字段中所读入的最多字符个数。

如果输入字段的字符个数少于 n,...scanf 函数读字段中的所有字符,然后是下一字段的格式指示符。

如果在读入给定宽度字符前遇到了空白字符或不可转换字符,已读的字符被~~转换~~存储,接着函数处理下一格式指示符。

不可转换字符是按给定格式不能转换的字符(如八进制格式下的 8 或 9,十六进制或十进制下的 J 或 K 等)。

宽度指示符 所存输入宽度的影响

n 读,转换 n 个字符,并存到当前地址参数中

9.输入大小与参数类型修饰符

输入大小修饰符(N 与 F)和参数类型修饰符(h 与 l)影响...scanf 函数对相应地址参数 arg 的解释。

F 和 N 覆盖 arg 缺省或声明的大小。

h 和 l 指明下面的输入数据使用什么类型(h=short,l=long)。输入数据将被转换成指定类型,输入数据的 arg 应指向对应大小的对象(%h 为 short,%l 为 long 或 double 对象)

修饰符	转换影响
F	覆盖缺省或声明的大小, arg 解释为远指针;
N	覆盖缺省或声明的大小,

	arg解释为近指针,
	不能在巨型模式中使用,
n	对d,i,o,u,x类型:转换输入为短整型,
	把它存在短整型对象中,
	对D,I,O,U,X类型:无影响,
	对e,f,c,s,n,p类型:无影响,
;	对d,i,o,u,x类型:转换输入为长整型, 存在长整型对象中,
	对e,f类型:转换输入为双精度型, 存在双精度对象中,
	对D,I,O,U,X类型:无影响,
	对c,s,n,p类型:无影响,

10. ...scanf 函数何时停止扫描

由于多种原因, ...scanf 函数在遇到正常字段结束符(空白字符)前, 可能停止扫描一特定字段或完全终止。

在下列情况下, ...scanf 函数停止扫描和存储当前字段, 而进入对下一字段的处理:

- 在格式指示符的百分号后出现赋值抑制字符(*), 当前输入字段被扫描但不存储
- 已读了 width 个字符(width 为宽度指示符, 它是格式指示中的一个十进制正整数)
- 在当前格式下不能转换下一输入字符(如在十进制格式下出现 A)。
- 输入字段中的下一个字符没有在搜集集中出现(或在相反搜集集中出现)
- 当出现以上情况, ...scanf 函数停止扫描当前输入字段的首字段时, 下一个字符假定未读, 被当作后一输入字段的首字符, 或当作输入后续读操作的首字符。

在下列情况下, ...scanf 终止扫描:

- 输入字段中的下一个字符与格式字符串中的相应空白字符冲突
- 输入字段中的下一字符为 EOF
- 格式字符串用完

如果在格式字符中出现一个不是格式指示部分的字符序列, 它必须与输入字段中的当前字符序列相匹配, ...scanf 函数扫描但不存储匹配字符。当出现冲突字符时, 仍保留在输入字段中, 就象没有读过一样。

返回值 所有的...scanf 函数都返回成功扫描、转换和存储的输入字段数, 被扫描但不存储的字段不算在内。

如果这些函数试图在文件末尾读(对于 sscanf 和 vsscanf, 为字符串尾), 返回 EOF。

如果没有字段被存储, 返回 0。

可移植性 函数 scanf, fscanf, sscanf 和 cscanf 适用于 UNIX 系统, 在 Kernighan 和 Richie 书中作了定义。

参 见 atof, getc, printf

源 程 序

```
#include <stdarg.h>
#include <stdio.h>
#include <_scanf.h>
#undef _ungetc /* remove the macro version */
int cdecl scanf(const char *fmt, ...)
{
    #pragma warn -sus
    return _scanner(
        (int) (* (void *)) fgetc,
        (void) (* (int ch, void *)) ungetc,
        stdin,
        fmt,
    );
}
```



```

        va_ptr
    );
#pragma warn .sus
}

```

函数名 `_scanner` - 读取格式化输入。 -- `scanner.cas`

用法 `int _scanner (int (*Get)(void *srceP),`
`void (*UnGet) (int ch, void *srceP),`
`void *srceP,`
`const char *formP,`
`va_list varPP)`

说明 所有 `scanf` 族函数的工作均由 `_scanner` 来做。

根据 `*formP` 给出的格式扫描 `*srceP` 并将结果放入 `*varPP`。

`Get` 和 `Unget` 函数允许将源重新定义为文件或其他串行字符流。它们可能是 `fget/ungetc` 或其它等价对, 处理数据流/串 `srceP`。

格式串的语法是:

```

format ::= ([isspace] [literal | '%%' | '%' conversion])*
conversion ::= ['*'] [width] ['l' | 'h'] [type]
width ::= number;
type ::= 'd'|'D'|'u'|'U'|'o'|'O'|'x'|'X'|'f'|'F'|'a'|
        'e'|'E'|'r'|'R'|'g'|'G'|'p'|'P'|'n'|'f'|'s'|'c'|'t'

```

源程序

```

#pragma inline
#include <asmrules.h>
#include <stdio.h>
#include <stdlib.h>
#include <scanf.h>
#define l asm
typedef enum
(
    _zz,          /* terminator */
    _ws,          /* space */
    _dc,          /* dont care */
    _pc,          /* percent */
    _su,          /* suppress */
    _nu,          /* numeral */
    _ch,          /* character */
    _de,          /* decimal */
    _un,          /* unsigned decimal - same as decimal */
    _in,          /* general int */
    _fl,          /* float */
    _ld,          /* long double */
    _ha,          /* half */
    _lo,          /* long */
    _oc,          /* octal */
    _st,          /* string */
    _sc,          /* scanset */
    _ct,          /* count of characters scanned */
    _he,          /* hexadecimal */
    _pt,          /* pointer */
    _ne,          /* near */
    _fa,          /* far */
)
charClass;
static const char scanCtype [128] =
{
    /* NUL SOH STX ETX EOT ENQ ACBSHEET LF VT FF CR SO SI */
    _zz, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc,
    /* DLE DC1 DC2 DC3 DC4 NAK SYN ETHERMANUB ESCFS GS RS US */
    _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc, _dc,

```

```

/*  dc, dc, dc, dc, dc, dc, dc, dc, dc, dc, dc, dc, dc, dc, dc, dc,
/*  SP, !, ", #, $, %, &, ' (, ), *, +, ,, -, ., /, : ; < = > ? [ \ ] ^ _ ` { | } ~ DEL */
/*  ws, dc, dc, dc, dc, pc, dc, dc, dc, dc, su, dc, dc, dc, dc, dc,
/*  0 1 2 3 4 5 6 7 8 9 : ; < = > ? [ \ ] ^ _ ` { | } ~ DEL */
/*  nu, nu, nu, nu, nu, nu, nu, nu, nu, nu, dc, dc, dc, dc, dc, dc,
/*  @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ ` { | } ~ DEL */
/*  dc, dc, dc, dc, dc, fl, fa, fl, ha, in, dc, dc, ld, dc, ne, oc,
/*  P Q R S T U V W X Y Z [ \ ] ^ _ ` { | } ~ DEL */
/*  dc, dc, dc, dc, dc, un, dc, dc, he, dc, dc, sc, dc, sc, dc, dc,
/*  a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ DEL */
/*  dc, dc, dc, ch, de, fl, fl, fl, ha, in, dc, dc, lo, dc, ct, oc,
/*  p q r s t u v w x y z { | } ~ DEL */
/*  pt, dc, dc, st, dc, un, dc, dc, he, dc, dc, dc, dc, dc, dc, dc, dc,
);
#pragma warn -use
#pragma warn -rvl
int _scanner ( int (*Get)(void *srceP),
               void (*UnGet) (int ch, void *srceP),
               void *srceP,
               const char *formP,
               va list varPP )
{
    char flags;
    int count = 0;
    int charCt = 0;
    int status;
    int width;
    char bitSet [32]; /* for scan sets */
    register SI, DI; /* prevent the compiler making its own usage */
    /******
    'C' equivalent of inline code( for documentation )
    *****/
    #if 0
    REG char a, b;
    REG short wP;
    REG char *cP;
    long IRes;
    long double ldRes;
    #error /* The C text is algorithm commentary, not tested source ! */
    /* It is provided to clarify the intent of the assembler. */
ssNEXT:
    if (\0 == (b = *(formP++)))
        return count; /* the normal end */
    if ((b != '%') || ('%' == (b = *(formP++)))
    {
        charCt ++;
        if ((a = Get (srceP)) == EOF)
            goto ssNextEOF;
        if (!(b & 0x80) && (_ws == scanCtype [b])) /* white space ? */
        {
            while (!(a & 0x80) && (_ws == scanCtype [a]))
            {
                charCt ++;
                if ((a = Get (srceP)) == EOF)
                    goto ssNextEOF;
            }
            UnGet (a, srceP);
            charCt --;
        }
        else /* literal match required */
            if (a != b)
                goto ssEND;
        goto ssNEXT;
    }
    /* if fall through to here then begin a conversion specification */
    #if LDATA
    flags = isFarPtr;
    #else
    flags = 0;
    #endif
    width = -1;
ssSwitch:

```

```

switch ((h & 0x80) ? _dc : scanCtype [b])
{
    case (_su) :    flags |= isSuppressed;
                   b = *(formP++);
                   goto ssSwitch;
    case (_ha) :    flags |= isHalf;
                   b = *(formP++);
                   goto ssSwitch;
    case (_lo) :    flags |= isLong;
                   b = *(formP++);
                   goto ssSwitch;
    case (_ld) :    flags |= isLongDouble;
                   b = *(formP++);
                   goto ssSwitch;
    case (_nu) :    width = (width < 0) ? b - '0' :
                           10 * width + b - '0';
                   b = *(formP++);
                   goto ssSwitch;
    case (_ne) :    flags &= ~isFarPtr;
                   b = *(formP++);
                   goto ssSwitch;
    case (_fa) :    flags |= isFarPtr;
                   b = *(formP++);
                   goto ssSwitch;
    case (_pt) :    goto ssPTR;
    case (_de) :    base = 10;
                   goto ssINT;
    case (_oc) :    base = 8;
                   goto ssINT;
    case (_he) :    base = 16;
                   goto ssINT;
    case (_in) :    base = 0;
                   goto ssINT;
    case (_ct) :    iRes = charCt;
                   if ((flags & isSuppressed) == 0)
                       goto ssPUTINT;
                   b = *(formP++);
                   goto ssSwitch;
    case (_fl) :    goto ssFLOAT;
    case (_st) :    goto ssTOKEN;
    case (_ch) :    goto ssCHAR;
    case (_sc) :    goto ssSCANSET;
    case (_dc) :    goto ssEND;
    default: /* never occurs. */;
}

ssINT:
    iRes = _scantol (Get, UnGet, srceP, base, width & 0x7FFF,
                    &charCt, &status);

ssPUTINT:
    if (('A' <= b) && (b <= 'Z'))
        flags |= isLong;
    if ((flags & isSuppressed) == 0)
    {
        if (flags & isLong)
            (long *) (*(varPP++)) = iRes;
        else if (flags & isHalf)
            (short *) (*(varPP++)) = iRes;
        else (int *) (*(varPP++)) = iRes;
        if (b != 'n')
            count++;
    }
    goto ssNEXT;

ssPTR:
    iRes = _scantol (Get, UnGet, srceP, 16, 4, &charCt, &status);
    if (status <= 0)
        goto ssEND;
    if (flags & isFarPtr)
    {
        if (':' != (b = Get (srceP)))
        {
            UnGet (b, srceP);
            goto ssEND;
        }
    }

```

```

        charCt ++;
        lRes = (lRes << 16) + _scantol (Get, UnGet, srceP, 16, 4,
                                         &charCt, &status);
        if (status <= 0)
            goto ssEND;
    }
    if ((flags & isSuppressed) == 0)
    {
        if (flags & isFarPtr)
            (long *) *(varPP++) = lRes;
        else
            (short *) *(varPP++) = lRes;
        count ++;
    }
    goto ssNEXT;
ssFLOAT:
    ldRes = scantod (Get, UnGet, srceP, width & 0x7FFF, &charCt, &status);
    if ((flags & isSuppressed) == 0)
    {
        if (flags & isLong)
            (double *) *(varPP++) = ldRes;
        else if (flags & isLongDouble)
            (long double *) *(varPP++) = ldRes;
        else
            (float *) *(varPP++) = ldRes;
        count ++;
    }
    goto ssNEXT;
ssTOKEN:
    while (_ws == scanCtype [a = Get (srceP)])
        charCt ++;
    charCt ++;
    if (EOF == a)
        goto ssEND;
    if ((flags & isSuppressed) == 0)
    {
        cP = *(varPP++);
        count ++;
    }
    while (_ws < scanCtype [a]) /* \0 or white space will terminate */
    {
        if ((flags & isSuppressed) == 0)
            *(cP++) = a;
        charCt ++;
        a = Get (srceP);
    }
    if ((flags & isSuppressed) == 0)
        *cP = \0;
    goto ssNEXT;
ssCHAR:
    if (width < 0)
        width = 1;
    if (width)
    {
        charCt ++;
        if ((a = Get (srceP)) == EOF)
            goto ssEOF;
    }
    if ((flags & isSuppressed) == 0)
    {
        cP = *(varPP++);
        count ++;
    }
    if (width)
    do
    {
        if ((flags & isSuppressed) == 0)
            *(cP++) = a;
        if (width --)
            break;
        charCt ++;
        if (EOF == (a = Get (srceP)))
            break;
    }
    goto ssNEXT;
ssSCANSET:
    /* scanset ::= '[' [ '^' ] [ '[' ] [ nonFinalSet ] ']' */

```

```

wP = & bitSet;
for (i = 16; *(wP++) = 0; i--);
exclude = false;
if ('^' == (a = *(formP++)))
{
    exclude = true;
    a = *(formP++);
}
for (;;) /* fill in the bit set */
{
    b = a;
    if (0 == a) /* unexpected end of format */
        goto ssEND;
    bitSet [a] = true;
    if (')' == (a = *(formP++)))
        break;
    if (('.' == a) && (b < *formP) && (']' != *formP))
    {
        a = *(formP++);
        while ( ++b < a)
            bitSet [b] = true;
    }
}
if (width = -1)
    width = 0x7FFF;
if (width)
{
    charCt ++;
    if ((a = Get (srceP)) == EOF)
        goto ssEOF;
}
if ((flags & isSuppressed) == 0)
{
    cP = *(varPP++);
    count ++;
}
while ((--width >= 0) && (exclude != (bitSet [a])) && (EOF != a))
{
    if ((flags & isSuppressed) == 0)
        *(cP++) = a;
    charCt ++;
    a = Get (srceP);
}
if (width >= 0)
{
    UnGet (a, srceP);
    charCt --;
}
if ((flags & isSuppressed) == 0)
    *(cP++) = '\0';
goto ssNEXT;
ssEND:
return (count) ? count : -1;
#endif
/***** End of C documentation *****/
goto RealCodeStart;
scn_Label_it:
/*-----Start of Local nested PROCs -----*/
I SCANNEXTARGPTR MACRO /* Note, Preprocessor sees this before TASM! */
I call scn_NextArgPtr
#ifdef HUGE
I jnc $+5
I jmp scn_END /* Can't scan NEAR w/HUGE model */
#endif
I ENDM
I scn_NextArgPtr PROC NEAR
I LES di, varPP
I test BY0 (flags), isFarPtr
I jz scn_nextNear
I les di, ES [di] /* ES: [di] = & result */
I add W0 (varPP), 4
#ifdef HUGE
I cld /* Clear carry bit */
#endif
I ret
scn_nextNear:

```

```

#ifdef _HUGE_
l stc /* Set carry bit */
#else
l mov di, ES_ [di] /* [di] = & DS: result */
l push ds
l pop es
l add W0 (varPP), 2
#endif
l ret
l scn_NextArgPtr ENDP
/*-----End of Local nested PROCs -----*/
RealCodeStart :
l push ES
l cld
scn_NEXT:
l mov si, formP
scn_nextChar:
#ifdef LDATA
l mov ES, W1 (formP)
#endif
l lods BYTE (ES_ [si]) /* *(formP++) */
l or al, al
l jz scn_respondJmp /* '\0' terminates the format string */
l cmp al, '%' /* conversion specs begin with '%' */
l je scn_CONV /* but "%%" returns to here */
scn_percent:
l cbw
l xchg di, ax
charCt ++;
Get (srceP);
l or ax, ax
l jl scn_EOFJmp
l or di, di
l js scn_mustMatch
l cmp BYTE (scanCtype [di]), _ws /* does format specify space ? */
l jne scn_mustMatch
scn_matchSpace:
l xchg ax, bx /* a format space matches any */
l or bl, bl /* number of source spaces */
l js scn_spaceEnded
l cmp BYTE (scanCtype [bx]), _ws
l jne scn_spaceEnded
charCt ++;
Get (srceP);
l or ax, ax
l jg scn_matchSpace
scn_EOFJmp:
l jmp scn_EOF
scn_spaceEnded:
UnGet (_BX, srceP); /* part of the next field */
charCt --;
l jmp short scn_nextChar
scn_mustMatch: /* non-space format characters must be */
l cmp ax, di /* matched in the input source. */
l je scn_nextChar
UnGet (_AX, srceP); /* part of the next field */
charCt --;
goto scn_END;
/* Jump via here in the normal case of end of format string. */
scn_respondJmp:
goto scn_respond;
/* If arrive here then a conversion specification has been entered. */
scn_CONV:
l mov W0 (width), -1
#ifdef LDATA
l mov ES, W1 (formP)
l mov BYTE (flags), isFarPtr
#else
l mov BYTE (flags), 0

```

```

#endif
scn_convNext:
l     lods     BY0 (ES_ [si])          /* *(formP++) */
l     cbw
l     mov     W0 (formP), si          /* remember the position */
l     xchg    di, ax
l     or      di, di
l     jl      scn_anyOther
l     mov     bl, _scanCtype [di]
l     xor     bh, bh
l     switch  ((charClass) _BX)
    {
case (_pc):
    goto scn_percent;          /* "%%" is literal '%' */
case (_zz):
    /* zero */
case (_dc):
    /* "don't care" came in spot where we 'care' */
case (_ws):
    /* unwanted space occurred. */
scn_anyOther:
    goto scn_END; /* any invalid specifier causes abrupt end. */
case (_su):
    /* Suppressed fields are scanned */
l     or      BY0 (flags), isSuppressed /* but the result is not stored. */
l     jmp     short scn_convNext
case (_nu):
    /* Scan widths set limits on field sizes. */
l     sub     di, '0'
l     xchg    width, di
l     or      di, di
l     jl      scn_convNext
l     mov     ax, 10
l     mul     di
l     add     width, ax
l     jmp     short scn_convNext
case (_ld):
    /* The LongDouble flag is used */
l     or      BY0 (flags), isLongDouble /* scanning for long doubles, */
l     jmp     short scn_convNext        /* nothing else. */
case (_lo):
    /* The "long" flag is used when */
l     or      BY0 (flags), isLong      /* scanning for long integers */
l     jmp     short scn_convNext        /* or doubles. */
case (_ha):
    /* The "half" flag is used when */
l     or      BY0 (flags), isHalf      /* scanning for short ints */
l     goto    scn_convNext;            /* or floats */
case (_ne):
    /* The "near" flag is used */
l     and     BY0 (flags), NOT isFarPtr /* when scanning for */
l     goto    scn_convNext;            /* 16-bit pointers */
case (_fa):
    /* The "far" flag is used */
l     or      BY0 (flags), isFarPtr    /* when scanning for */
l     goto    scn_convNext;            /* 32-bit pointers */
case (_ct):
    /* The count of source chars read */
l     mov     ax, charCt                /* characters used. */
l     sub     dx, dx
l     test    BY0 (flags), isSuppressed
l     jz      scn_PUTINT
l     goto    scn_convNext;
case (_oc):
l     mov     si, 8                      /* radix = 8 */
l     jmp     short scn_INT
case (_un):
case (_de):
l     mov     si, 10                     /* radix = 10 */
l     jmp     short scn_INT
case (_he):
l     mov     si, 16                     /* radix = 16 */
l     jmp     short scn_INT
case (_in):
l     mov     si, 0                      /* source syntax will decide radix */
/* Arrive here if an integer is expected. Signed or unsigned are treated
   similarly, the _scantol routine will handle either and there is no method
   (or good purpose) to complaining about a '-' if the user enters it. */
scn_INT:
scn_UINT:

```

```

        if ((DI & 0x20) == 0) flags |= isLong; /* bit 5 is set if lower case */
        _scantol (Get, UnGet, srceP, _S1, width & 0x7FFF, &charCt, &status);
1      cmp     W0 (status), 0
1      jle     scn_intEnd
1      test    BY0 (flags), isSuppressed
1      jnz     scn_intUpdated
1      inc     W0 (count)
scn_PUTINT:                                /* from %n specifications */
1      SCANNEXTARGPOINTER /* This is a MACRO! */
/* On the iAPX-86 family int == short, so we can ignore "isHalf". */
scn_intStos:
1      stosw
1      test    BY0 (flags), isLong
1      jz      scn_intUpdated
1      xchg    ax, dx
1      stosw
scn_intUpdated:
        goto   scn_NEXT;
scn_intEnd:
1      jl      scn_intEOF
        goto   scn_END;
scn_intEOF:
        goto   scn_EOF;
/* Pointer formats are in Intel style, either hhhh (DS default) or
   hhhh:hhhh for far. */
case (_pt):
        SimLocalCall
1      jmp     scn_SkipSpace /* CX zapped, result in AL */
        UnGet (_AX, srceP);
        charCt --;
1      and     W0 (width), 7FFFh
        SimLocalCall
1      jmp     scn_InHex4 /* CX zapped, next char in AX, */
                        /* numeric result in DX */
1      push    dx /* save the MSW of pointer */
1      cmp     al, ' '
1      je      scn_ptrLSW
1      or      ax, ax
1      jle     scn_noLookAhead
        UnGet (_AX, srceP);
        charCt --;
scn_noLookAhead:
1      pop     dx /* retrieve, use as LSW */
#ifdef HUGE
1      jmp     short scn_ptrEndJmp
#else
1      mov     bx, DS
1      jmp     short scn_pointerReady
#endif
scn_ptrLSW:
        SimLocalCall
1      jmp     scn_InHex4 /* CX zapped, next char in AX, */
                        /* numeric result in DX */
1      pop     bx /* retrieve the MSW */
1      or      ax, ax
1      jle     scn_notAhead
1      push    dx
1      push    bx
        UnGet (_AX, srceP);
        charCt --;
1      pop     bx
1      pop     dx
scn_notAhead:
scn_pointerReady:
1      test    BY0 (flags), isSuppressed
1      jnz     scn_ptrUpdated
1      SCANNEXTARGPOINTER /* This is a MACRO! */
1      inc     W0 (count)
1      xchg    ax, dx

```



```

I      stosw
I      xchg  ax, bx
I      stosw
scn_ptrUpdated:
I      goto  scn_NEXT;
scn_ptrEnd:
I      jl     scn_jmpEOF
scn_ptrEndJmp:
I      goto  scn_END;
scn_jmpEOF:
I      jmp    scn_EOF
/*-----
'e', 'r', 'g', 'E' and 'G' formats are all equivalent for input
since _scantod and _scantoLd recognize all variants.
-----*/
case ( fl):
#if LDATA
I      push   SS
#endif
I      lea    ax, status
I      push   ax /* , &status */
#if LDATA
I      push   SS
#endif
I      lea    ax, charCt
I      push   ax /* , &charCt */
I      mov    ax, 07FFFh
I      and    ax, width /* , 0x7FFF & width */
I      push   ax
#if LDATA
I      push   W1 (srceP)
#endif
I      push   W0 (srceP) /* , srceP */
#if LPROC
I      push   W1 (UnGet)
#endif
I      push   W0 (UnGet) /* , UnGet */
#if LPROC
I      push   W1 (Get)
#endif
I      push   W0 (Get) /* , Get */
I      call   EXTPROC ( scantod) /* ST(0) = scantod ( . */
I      add    sp, (3 * dPtrSize) + 2 + (2 * cPtrSize)
I      cmp    W0 (status), 0
I      jle    scn_endFloat
I      if (!(flags & isSuppressed))
I      {
I          SCANNEXTARGPOINTER /* This is a MACRO! */
I          inc    W0 (count)
I          test   BY0 (flags), isLong /* is it 'double' */
I          jz     test_LongDouble
I          mov    ax, isLong
I          jmp    short push_type
test_LongDouble:
I          test   BY0 (flags), isLongDouble /* is it 'long double' */
I          jz     its_default_float
I          mov    ax, isLongDouble
I          jmp    short push_type
its_default_float:
I          xor    ax, ax /* default is 'float' */
push_type:
I      push    ax
I      #if LDATA
I          push   es
I      #endif
I          push   di
I          call   EXTPROC (_scaarslt)
I          add    sp, dPtrSize + 2
I          goto  scn_NEXT; /* This item is complete*/

```

```

    }
scn_popFloat:      /* if suppressed then discard result of _scantod. */
I      call      EXTPROC (_scanpop) /* pop stack */
scn_UpdatedReal:
    goto scn_NEXT;
scn_endFloat:      /* if failed then discard result of _scantod. */
I      call      EXTPROC (_scanpop) /* pop stack */
I      jl      scn_impEOF
    goto scn_END;
/*****
The 's' conversion specifies to take a "token" from the source.
The token is the next contiguous group of non-space characters.
*****/
case (_st):
scn_untilToken:
    SimLocalCall
I      jmp      scn_SkipSpace /* CX zapped, result in AL */
scn_maybeToken:
I      test     BY0 (flags), isSuppressed
I      jnz      scn_tokenWidth
I      SCANNEXTARGPOINTER /* This is a MACRO! */
I      inc      W0 (count)
scn_tokenWidth:
I      and      width, 7FFFh /* default width becomes 7FFFh */
I      jz      scn_tokenEnd
scn_whileToken:
I      test     BY0 (flags), isSuppressed
I      jnz      scn_tokenNextCh
I      stosb
scn_tokenNextCh:
    charCt ++;
I      push     ES
    Get (srceP);
I      pop      ES
I      or       ax, ax
I      jle      scn_tokenEnd /* end if \0 or EOF */
I      or       al, al
I      js       scn_isToken
I      xchg     bx, ax
I      cmp      scanCtype [bx], _ws
I      xchg     ax, bx
I      jng      scn_tokenEnd /* end if space */
scn_isToken:
I      dec      W0 (width)
I      jg       scn_whileToken /* width limits loop count */
scn_tokenEnd:
I      push     ES
    UnGet (_AX, srceP);
I      pop      ES
    charCt --;
I      test     BY0 (flags), isSuppressed
I      jnz      scn_tokenUpdated
I      mov      al, 0 /* terminate result token string. */
I      stosb
scn_tokenUpdated:
    goto scn_NEXT;
/* The 'c' option captures a literal character array. Leading and embedded
space characters are taken, not skipped. The array size must be filled:
if too few characters are in the source then the conversion fails and is
not counted. */
case (_ch):
I      test     BY0 (flags), isSuppressed
I      jne      scn_checkWidth
I      SCANNEXTARGPOINTER /* This is a MACRO! */
scn_checkWidth:
I      mov      si, width /* if width was -1 (default), then */
I      or       si, si /* set it to 1. Note that a zero */
I      jnl      scn_charWidened /* width is valid (consider how */
I      mov      si, 1 /* an '*' width might be used). */

```

```

scn_charWidened:
I      jz      scn_charEnd          /* skip if user set a zero width */
scn_charLoop:
I      charCt ++;
I      push    ES
I      Get (srcP);
I      pop     ES
I      test    BY0 (flags), isSuppressed
I      jne     scn_charNoPut
I      stosb
scn_charNoPut:
I      or      ax, ax
I      jl      scn_charEOF
I      dec     si                    /* width */
I      jg      scn_charLoop
scn_charEnd:
I      test    BY0 (flags), isSuppressed
I      jne     scn_charNext
I      inc     W0 (count)
scn_charNext:
I      goto    scn_NEXT;
scn_charEOF:
I      goto    scn_EOF;          /* source was incomplete */
case ( sc):
#if LDATA
I      push    ES
#endif
I      sub     ax, ax
I      cld
I      push    SS
I      pop     ES
I      lea     di, bitSet
I      mov     cx, 16
I      rep stosw                  /* clear 256 bits. */
#if LDATA
I      pop     ES
#endif
I      lods    BY0 (ES_ [si])      /* *(formP++) */
I      and     BY0 (flags), NOT isExclusive
I      cmp     al, '^'
I      jne     scn_scanInc
I      or      BY0 (flags), isExclusive
I      lods    BY0 (ES_ [si])      /* *(formP++) */
scn_scanInc:
I      mov     ah, 0
scn_scanSetBit:
I      mov     dl, al
I      mov     di, ax
I      mov     cl, 3
I      shr     di, cl
I      mov     cx, 107h
I      and     cl, di
I      shl     ch, cl
I      or      bitSet [di], ch
scn_setNext:
I      lods    BY0 (ES_ [si])      /* *(formP++) */
I      cmp     al, 0              /* unexpected end of format ? */
I      jz      scn_scanOpen
I      cmp     al, '^'           /* normal end of scan set specification */
I      je      scn_scanBounded
I      cmp     al, '^'           /* possible range specification */
I      jne     scn_scanSetBit
I      cmp     dl, ES_ [si]
I      ja      scn_scanSetBit
I      cmp     BY0 (ES_ [si]), '^'
I      je      scn_scanSetBit
/* If arrived here then a range has been specified, and note that the first
   bit of the range has already been set. */
I      lods    BY0 (ES_ [si])      /* *(formP++) */

```

```

:      sub    al, dl          /* AL is count of bits needed. */
:      je     scn_setNext
:      add    dl, al          /* DL = (A + Z-A) = Z, final char. */
scn_setRange:
:      rol    ch, 1          /* next bit position is in .. */
:      adc    di, 0          /* .. next byte if wrap-around. */
:      or     bitSet [di], ch
:      dec    al
:      jnz    scn_setRange
:      jmp     short  scn_setNext
scn_scanOpen:                /* scan set was not written correctly */
:      goto   scn_END;       /* abandon the conversion */
scn_scanBounded:            /* the closing ']' has been found. */
:      mov     formP, si      /* remember formP next position */
:      and     W0 (width), 7FFFh /* convert default -1 to large positive */
:      mov     si, width
:      test    BY0 (flags), isSuppressed /* if suppressed, then just skip */
:      jne     scn_scanLoop   /* input, generate no result */
:      SCANNEXTARGPOINTER    /* This is a MACRO! */
scn_scanLoop:
:      dec     si
:      jl      scn_scanLimited
:      charCt ++;
:      push    ES
:      Get (srceP);
:      pop     ES
:      or      ax, ax
:      jl      scn_scanEOF
:      xchg    si, ax
:      mov     bx, si
:      mov     cl, 3          /* calculate bit equivalent of char */
:      shr     si, cl
:      mov     cx, 107h
:      and     cl, bl
:      shl     ch, cl
:      test    ch, bitSet [si] /* is the character in the scan set ? */
:      xchg    ax, si
:      xchg    ax, bx
:      jz      scn_scanNotIn
/* If arrived here then the char was in the scan set. */
:      test    BY0 (flags), isExclusive /* exclusive scan ? */
:      jz      scn_scanAccept
:      jmp     short  scn_scanTerminate
/* If arrived here then the char was not in the scan set. */
scn_scanNotIn:
:      test    BY0 (flags), isExclusive /* exclusive scan ? */
:      jz      scn_scanTerminate
/* If arrived here, then AL holds an acceptable character */
scn_scanAccept:
:      test    BY0 (flags), isSuppressed /* if suppressed, then just skip */
:      jne     scn_scanLoop   /* input, generate no result. */
:      stosb                                /* move character to result string */
:      jmp     short  scn_scanLoop
:
/* If arrived here then the end of the scanned token has been found. */
scn_scanTerminate:
:      push    ES
:      UnGet ( AX, srceP); /* unget the terminator */
:      pop     ES
:      charCt --;
:      inc     si
:      cmp     si, width
:      jnl     scn_scanEND    /* No match at all was seen */
/* If arrived here then the maximum width was hit or scan is complete. */
scn_scanLimited:
:      test    BY0 (flags), isSuppressed /* if suppressed, then just skip */
:      jne     scn_scanUpdated /* input, generate no result */
:      inc     W0 (count)
scn_scanEND:

```

```

I      mov     al, 0
I      stosb
scn_scanUpdated:
I      goto    scn_NEXT;
scn_scanEOF:
I      inc     si
I      cmp     si, width
I      jnl     scn_EOF /* input failed before token */
I      test    BY0 (flags), isSuppressed /* if suppressed, just skip */
I      jne     scn_EOF
I      mov     al, 0
I      stosb
I      inc     W0 (count)
/*      jmp     scn_EOF */
}
/*      *** End of Switch *** */
/* Input failure before a conversion succeeds. Give a count of
the number of input fields successfully taken: if failed before
any input accepted, return EOF; */
scn_EOF:
I      UnGet (EOF, srceP);
I      cmp     W0 (count), 1 /* generates carry if count = 0 */
I      shb     W0 (count), 0 /* so had count becomes -1 */
/* Arrive here for conversion failures. */
scn_END:
/* If arrived here then the scan ended gracefully. Return a count of input
fields accepted. */
scn_respond:
scn_exit:
I      pop     ES
I      return count;
/* --- Local Subroutines --- */
/* --- SkipSpace --- skip spaces at beginning of a spec */
scn_SkipSpace:
I      charCt ++;
I      Get (srceP);
I      or      ax, ax
I      jle     scn_skipEOF /* \0 or EOF */
I      or      al, al
I      js      scn_skipEnd
I      xchg    bx, ax
I      cmp     scanCtype [bx], _ws /* white space ? */
I      xchg    ax, bx
I      je      scn_SkipSpace
scn_skipEnd:
I      pop     cx /* get return address */
I      add     cx, 3 /* avoid the 3-byte jump */
I      jmp     cx /* RETNEAR */
scn_skipEOF:
I      jz      scn_skipEnd
I      pop     cx /* discard normal return */
I      jmp     short scn_EOF /* HUGE has no default DS */
/* --- InHex4--- Collect up to 4 hex digits
result: CX zapped, next char in AX, numeric result in DX. */
scn_InHex4:
I      sub     dx, dx
I      mov     cx, 4
scn_h4loop:
I      dec     W0 (width)
I      jl      scn_h4limited
I      push    dx
I      push    cx
I      charCt ++;
I      Get (srceP);
I      pop     cx
I      pop     dx
I      or      ax, ax
I      jle     scn_h4end /* \0 or EOF */
I      dec     cl
I      jl      scn_h4end /* maximum of 4 digits */

```

```

1      mov     ch, al
1      sub     ch, '0'
1      jb      scn_h4end
1      cmp     ch, '10'
1      jb      scn_h4isDigit
1      sub     ch, 'A' - '0'
1      jb      scn_h4end
1      cmp     ch, '6'
1      jb      scn_h4isHex
1      sub     ch, 'a' - 'A'
1      jb      scn_h4end
1      cmp     ch, '6'
1      jnb     scn_h4end
scn_h4isHex:
1      add     ch, 10
scn_h4isDigit:
1      shl     dx, 1
1      shl     dx, 1
1      shl     dx, 1
1      shl     dx, 1
1      add     dl, ch
1      jmp     scn_h4loop
scn_h4limited:
1      sub     ax, ax          /* no lookahead          */
scn_h4end:
1      cmp     cl, 4
1      je      scn_h4eof
1      pop     cx              /* get return address    */
1      add     cx, 3           /* avoid the 3-byte jump */
1      jmp     cx              /* RETNEAR               */
scn_h4eof:
1      pop     cx              /* discard normal return */
1      jmp     scn_EOF        /* reject input          */
}
#pragma warn .use
#pragma warn .rvl

```

函数名 **scanpop** - 在转换错误之后清理堆栈。 -- scantod.cas

用法 **void _scanpop(void);**

说明 该函数用于在 **_scanner** 函数中产生转换错误之后清理堆栈。

源程序

```

static void _scanpop()
{
asm      FSTP     ST(0)    /* pop math coprocessor stack */
}

```

函数名 **scanrslt** - 获取转换结果。 -- scantod.cas

用法 **void _scanrslt(double *rsltP, int rsltType);**

说明 该函数用于获取 **_scanner** 函数中的错误结果。

源程序

```

static void _scanrslt(void *rsltP, int rsltType)
{
    long double temp;
asm      FSTP     LONGDOUBLE (temp)
    if (rsltType & isLong)
        *(double *)rsltP = _ldtrunc(DBL, temp, HUGE_VAL);
    else if (rsltType & isLongDouble)
        *(long double *)rsltP = temp;
    else
        *(float *)rsltP = __ldtrunc(FLT, temp, 1./0.);
}

```

注 释 上述函数都是 scanf 族函数必须使用的。

函 数 名 scantod - 将字符串转换为浮点数。 - scantod.cas

用 法 long double _scantod (int (*Get) (void *srceP),
void (*UnGet) (int ch, void *srceP),
const void *srceP,
int width,
int *countP,
int *statusP)

原型文件 _scanf.h

说 明 将字符串转换为双精度实数。字符串的语法是:

```
float ::= [isspace]* [sign] [realnum] [exponent]
isspace ::= as per <ctype.h>:isspace
realnum ::= {digit [digit]* ['.' [digit]* ]} |
           {'.' digit [digit]*}
exponent ::= 'e'|'E' [sign] digit [digit]*
```

SrcP 指向待扫描的对象,例如,它可以是 FILE*。函数 GET 或 Unget 通过对 SrcP 获取字符,并可能根据 LR(1)扫描规则替换字符。

数字必须是十进制的。

width 是可以接受的数字位数的限制。如果有符号,也包括在内。但不包括前面的空格。

返回给调用程序的计数值是所消耗的字符个数,包括数前面的空格(即使没找到数)。它是加到已存在的计数值中。

在转换开始之前就遇到 EOF,则返回的状态是 EOF;如果在出现数字之前遇到其他字符则返回的状态是 0;如果转换正确则返回的状态是 1;如果出现上溢或下溢则返回 2。

如果源字符串不是合法的浮点数,则结果值为 0,并且源字符串中的下一个字符不能成为数的一部分的第一个字符。如果该数太大或太小,则结果为 HUGE_VAL 或 0。

方 法: 转换分两步进行。首先捕获串中的小数部分和指数部分。

小数部分可以是一个 63bit 的无符号整数(18 位精度),并且具有单独的符号。超出 18 位的数字则被截掉。

指数是一个两进制格式的短整数,并被调整到注意小数部分小数点的位置,这样小数部分作为带有小数点的整数按右对齐进行规范化。

捕获了小数部分和指数部分后,第二步是组合它们。这是通过以下公式进行的:

结果 = $10^{(\text{指数部分})} \times \text{小数部分} \times \text{符号}$

如果结果上溢,则返回 ±HUGE。如果结果下溢,则返回 0。

这里并不使用 INDP-87,虽然当用户安装了协处理器时这是最佳的。为了寻求平衡,该过程有策略地,但不频繁地使用,因为用软件模拟速度非常慢。

下图可能有助于理解变量之间的关系:

0000123456789012345.098765432109876E+99

-----十进制数----->|
|-----*数字*>| 不包括“.”

如果“.”是左边第一位则十进制数被当作负的。数字总是正的,除非没有非零数字。

返 回 值 返回输入串的连接值。

源 程 序

```
#pragma inline
#include <asmrules.h>
```

```

#include < scanf.h>
#include < ctype.h>
#include < math.h>
#include < stdlib.h>
/* Internal RTL function to perform double/float truncations. */
#define FLT 0
#define DBL 1
double pascal ldtrunc(int flag, long double x, double xhuge);
/* +/- infinity, +/- NAN */
static const float INF = 1.0/0.0;
static const float INFM = -(1.0/0.0);
static const float NAN = 0.0/0.0;
static const float NANM = -(0.0/0.0);
#pragma warn -rvl
#pragma warn -use
static long double _scantod (
    int (*Get) (void *srceP),
    void (*UnGet) (int ch, void *srceP),
    const void *srceP,
    int width,
    int *countP,
    int *statusP )
(
    int decimals; /* register SI = 0x8000 */
    int digits; /* register DI = -2 */
    int exponent;
    char sign = 0;
    char FirstDigit = 1;
    char saw sign = 0;
    char expSign = 0;
    char ExpOfFlow = 0;
    int ct = 0;
    int status = 1;
    long double frac = 0.0;
    register SI, DI; /* prevent compiler usage */
asm push ES
asm mov si, 8000h
asm mov di, -2
/* Skip leading spaces on the input numeral. */
std_nextBlank:
    ct ++;
    Get (srceP);
asm or ax, ax
asm jnl not_instantEOF /* No EOF the first time */
asm jnlp std_EOF /* EOF happened first thing */
not_instantEOF:
asm cbw
asm xchg bx, ax
asm test bl, 80h
asm jnz std_notSpace
#if _HUGE_
asm mov ax, seg _ctype
asm mov ES, ax
asm test BYTE (ES: _ctype [bx+1]), _IS_SP
#else
asm test BYTE (_ctype [bx+1]), _IS_SP
#endif
asm jnz std_nextBlank
std_notSpace:
asm xchg ax, bx
asm dec W0 (width)
asm jl std_fractionLimited
/* Is the numeral preceded by a sign ? */
asm cmp al, '+'
asm je std_signSeen
asm cmp al, '-'
asm jne std_fracChar /* AL must hold a fraction character. */
asm inc BYTE (sign) /* set flag to true == negative */
std_signSeen:
    saw_sign ++;

```



```

std_fracLoop:                                /* Pick up the next character of the fraction. */
asm      dec      W0 (width)
asm      jl      std_fractionLimited
        ct ++;
        Get (srcP);
/*-----
We need to check for the special cases where +INF -INF +NAN -NAN
might be specified.
-----*/
asm      cmp      BY0 (FirstDigit), 1
asm      jne      std_fracChar                /* Its not 1st char, continue */
asm      cmp      BY0 (saw sign), 0
asm      je       std_fracChar                /* There was no sign, continue */
asm      cmp      al, 'I'
asm      je       relPossibleINF              /* Maybe we have +/-INF */
asm      cmp      al, 'N'
asm      je       relPossibleNAN              /* Maybe we have +/-NAN */
asm      jmp      short std_fracChar          /* Its not a special case */
relPossibleINF:
asm      jmp      PossibleINF;                /* far jmp within relative range*/
relPossibleNAN:
asm      jmp      PossibleNAN;                /* far jmp within relative range*/
std_fracChar:
asm      mov      BY0 (FirstDigit), 0
asm      cmp      al, '.'                      /* Watch for decimal points */
asm      je       std_fracPoint
asm      cmp      al, 'y'
asm      ja       std_fracEndJmp              /* All other non-numeric characters .. */
asm      cmp      al, '0'
asm      jb       std_fracEndJmp              /* .. are fraction terminators. */
asm      sub      al, '0'                      /* convert digit to equivalent number. */
asm      cbw
/* Keep a count of the digits seen. */
asm      inc      di
asm      jg      std_notFirst                  /* was it the first digit ? */
/* The first digit begins the fraction. Leading zeros are noted by setting
digits -1, so that the fraction syntax is valid if no other digits
are seen, but following digits will still be treated as "firsts".
Leading non-zero digits cause digits to be set to 1. */
asm      mov      frac [0], al
asm      mov      di, 1
asm      or      al, al
asm      jnz      std_fracLoop
asm      neg      di
asm      cmp      si, 8000h                    /* has decimal point been seen ? */
asm      je       std_fracLoop
asm      dec      si                          /* if yes, move it to the left. */
asm      jmp      short std_fracLoop
/* Arrive here when fraction is width-limited but valid. */
std_fractionLimited:
asm      mov      al, 'e'                      /* Behave as if exponent started. */
jmp_to_fracEnd:                               /* Label within relative range */
asm      jmp      std_fracEnd                  /* Width will limit exponent, too. */
/* Error action placed here for short jump range. */
std_EOF:
        status = -1;
asm      jmp      short std_noResult
std_fracEndJmp:                               /* extend jump range */
asm      jmp      std_fracEnd
/* A decimal point has been seen */
std_fracPoint:
asm      cmp      si, 8000h                    /* Has a previous decimal point been seen ? */
asm      jne      jmp_to_fracEnd              /* If so, the fraction is terminated. */
asm      sub      si, si                      /* result if '.' before any digit */
asm      or      di, di
asm      jng      std_fracLoop
asm      mov      si, di                      /* decimals = digits */
asm      jmp      short std_fracLoop
/* If a digit is seen, then multiply the existing fraction by 10 and

```

add in the new digit. The special case of the first 5 digits is treated separately for speed. */

```

std_notFirst:
asm    cmp     di, 5
asm    ja      std_beyond5
asm    xchg    bx, ax
asm    mov     ax, 10
asm    mul     W0 (frac)
asm    add     ax, bx
asm    adc     di, db
asm    mov     frac [0], ax
asm    mov     frac [2], dx
asm    jmp     std_fracLoop
/* Digits beyond the 6th are more rare in practice (even in 6-digit
numbers, 5 will be quick), so no further special cases are
justified. Beyond 18 digits, ignore the digit values but
keep scanning. */

```

```

std_beyond5:
asm    cmp     di, 18
asm    ja      jmp_frac_loop
asm    xchg    bx, ax
asm    mov     ax, 10
asm    mul     W0 (frac [6])
asm    mov     (frac [6]), ax
asm    mov     ax, 10
asm    mul     W0 (frac [4])
asm    mov     (frac [4]), ax
asm    push    dx
asm    mov     ax, 10
asm    mul     W0 (frac [2])
asm    mov     (frac [2]), ax
asm    push    dx
asm    mov     ax, 10
asm    mul     W0 (frac [0])
asm    add     ax, bx
asm    mov     (frac [0]), ax
asm    adc     (frac [2]), dx
asm    pop     dx
asm    adc     (frac [4]), dx
asm    pop     dx
asm    adc     (frac [6]), dx
jmp_frac_loop:
asm    jmp     std_fracLoop
/* error clauses placed here within short-jump range of whole routine.
Arrive here if an error occurred. */

```

```

std_noDigitSeen:
    status = 0;
std_noResult:
    if (width >= 0)
    {
        UnGet (_AX, srcP);
        ct --;
    }
asm    FLDZ                      /* and a zero numeric result. */
asm    jmp     std_end
/** end of error clauses. */
/* The fraction was ended. If it was valid, it must have had at least
one digit. AL must hold the character which terminated the fraction. */
std_fracEnd:
asm    cmp     di, -2
asm    jz      std_noDigitSeen
/* If no decimal point was seen, then the decimal is assumed to be at
the rightmost edge. */
asm    cmp     si, 8000h
asm    jne     std_exponent
asm    mov     si, di              /* decimals = digits */
/* Now we must gather the exponent. First, check for 'E' or 'e' to
introduce it, then if found gather the short integer. */
std_exponent:
asm    mov     digits, di

```

```

asm    mov    decimals, si
asm    sub    di, di          /* DI = exponent */
asm    cmp    al, 'E'
asm    je     std_present
asm    cmp    al, 'e'
asm    jne    std_combine
std_present:
asm    dec    W0 (width)
asm    jl     std_exponentLimited
        ct ++;
        Get (srceP);
asm    cmp    al, '+'
asm    je     std_expNext
asm    cmp    al, '-'          /* is exponent negative ? */
asm    jne    std_expGotNext
        expSign ++;
std_expNext:
asm    dec    W0 (width)
asm    jl     std_exponentLimited
        ct ++;
        Get (srceP);
std_expGotNext:          /* if no leading sign, must be leading digit. */
asm    cmp    al, '9'
asm    ja     std_combine
asm    sub    al, '0'
asm    jnb    std_expNonDigit
asm    cbw
/* The largest IEEE long doubles have exponents -4932 <= X <= +4932.
Numbers outside that range will be accepted as infinite or zero,
according to the sign of the exponent. */
std_expLoop:
asm    xchg    ax, di
asm    mov    dx, 10
asm    mul    dx
asm    add    di, ax          /* DI = exponent */
asm    cmp    di, 4932        /* The upper limit on exponents */
asm    jle    std_expNext
asm    xor    di, di          /* Exponent overflow, set flag */
asm    mov    BY0 (ExpOfLow), 1
asm    jmp    short std_expNext
std_expNonDigit:
asm    add    al, '0'          /* restore original terminator */

/* Arrive here when a valid syntax has been terminated.
AL must still contain the terminating character, unchanged.
std_combine:
    UnGet (_AX, srceP);
    ct--;
/* Arrive here with valid termination but no terminator to be pushed back.
std_exponentLimited:
asm    test    BY0 (expSign), 0FFH /* was the exponent signed ?
asm    jz      skip_neg
asm    neg     di
/* Normal stays normal, Infinity becomes 0 if exponent was hugely negative.
asm    neg     BY0 (ExpOfLow)
skip_neg:
/* The special case when digits = -1 occurs when the fraction is zero.
In that case, the result is always zero, whatever the exponent.
asm    mov     bx, digits
asm    or      bx, bx
asm    jnl     std_nonZero
asm    FLDZ
asm    jmp     std_end
/* Combine the decimal point position with the exponent. The exponent
begins with a value that reflects the position of the decimal point.
std_nonZero:
asm    mov     cx, decimals
asm    mov     ax, cx
asm    add     ax, di          /* 1.0E(decimals+exponent) = upper bound */

```

```

/*                                0.1E(decimals+exponent) = lower bound
Convert underflows to zero and overflows to ld HUGE_VAL. */
asm    cmp    BY0 (ExpOfLow), 1      /* big (+) exp -> ld HUGE_VAL */
asm    je     std_isInfinite
asm    cmp    BY0 (ExpOfLow), -1     /* big (-) exp -> 0 */
asm    jne    std_isNormal
std_isZero:
asm    FLDZ
asm    jmp     short status2
std_isInfinite:
/* - Make 'frac' a long double HUGE_VAL */
asm    mov     ax, -1
asm    mov     frac[0], ax
asm    mov     frac[2], ax
asm    mov     frac[4], ax
asm    mov     frac[6], ax
asm    mov     frac[8], 07FFEH
asm    FLD     LONGDOUBLE( frac )
status2:
asm    mov     W0 (status), 2
asm    jmp     std_end
std_isNormal:
/* For normal numbers multiply fraction * 10^exponent */
asm    mov     ax, bx
asm    cmp     bx, 18
asm    jna     std_usualPoint
asm    mov     bx, 18      /* a maximum of 18 digits are used */
std_usualPoint:
asm    add     ax, cx
asm    sub     cx, bx
std_beginExp:
/* CX = decimal point contribution to exponent */
/* DI = combined exponent */
asm    add     di, CX
asm    FILD    qword ptr (frac)
asm    mov     ax, di
asm    or      ax, ax
asm    jz      std_end     /* no multiply required if exponent is zero. */
asm    jnl     std_pow
asm    neg     ax
std_pow:
asm    push    ax
asm    call    EXTPROC (pow10) /* leaves result in iNDP-87 ST(0) TOS */
asm    pop     ax
asm    or      di, di
asm    jnl     std_expPlus
asm    FDIV    /* negative exponent --> 1 / 10^|exponent| */
asm    jmp     short std_end
std_expPlus:
asm    FMUL    /* combine the exponent with the fraction. */
std_end:
    if (sign)
    {
asm    FCHS    /* negate the result */
    }
std_returnPP: /* update *(suffixPP) with the next character's position. */
/* - Finally, of course, don't forget to return the converted number ! */
std_exit:
asm    LES     di, countP
asm    mov     bx, ct
asm    add     ES [di], bx
asm    LES     di, statusP
asm    mov     bx, status
asm    mov     ES [di], bx
asm    pop     ES
return;
PossibleINF:
    ct ++;
    Get (srceP);
asm    dec     W0 (width)
asm    jl      Didnt_pan_out
asm    cmp     al, 'N'

```


基数可以是 0，也可以是 2、36 的数。如果基数为 0，则根据 C 语言区分八、十、十六进制的规则从 8、10 或 16 中选择可能的基数。

如果基数 > 10，则字母 'A..Z' 构成合法数字的扩充集。

如果基数非法或找不到数，则结果值为 0，并且输入串中的数字不变。

width 是可以接受的数字位数的限制。如果有符号，也包括在内。但不包括前面的空格。

返回给调用程序的计数值是所消耗的字符个数，包括数前面的空格（即使没找到数）。它是加到已存在的计数值中。

如果在转换开始之前就遇到 EOF，则返回的状态是 EOF；如果在出现数字之前遇到其他字符则返回的状态是 0；如果转换正确则返回的状态是 1；如果出现上溢则返回 2。

源 程 序

```
#pragma warn -rvi
#pragma warn -use
long _scantol (int (*Get) (void * srceP),
               void (*UnGet) (int ch, void * srceP),
               const void *srceP,
               int radix,
               int width,
               int *countP,
               int *statusP )
{
    char
    int
    int
    register
    sign = 0;
    ct = 0;
    status = 1;
    SI, DI; /* prevent the compiler making its own usage */
asm
/*      push
/*      First skip over any white-space prefix.
/*      avoid assuming DS
/*      */
asm
/*      mov
/*      di, 1 + offset (ES, _ctype)
stl_skipSpace:
    ct ++;
    Get (srceP);
asm
    or
asm
    jl
asm
    cbw
asm
    xchg
asm
    test
asm
    jnz
#ifdef __HUGE__
asm
    mov
asm
    mov
#endif
asm
    test
asm
    jnz
stl_notSpace:
asm
    xchg
asm
    dec
asm
    jl
asm
/* next check for an optional negative sign. */
asm
    cmp
asm
    je
asm
    cmp
asm
    jne
    sign ++;
stl_signSeen:
asm
    dec
asm
    jl
    ct ++;
    Get (srceP);
asm
    or
asm
    jl
stl_signed:
asm
    sub
asm
    mov
asm
    mov
    si, si /* DI:SI hold the result.
    di, si /* default result is zero.
    cx, radix
```

```

asm          jcxz    stl_autoRadix
asm          cmp     cx, 36
asm          ja      stl_badRadix
asm          cmp     cl, 2
asm          jnb     stl_badRadix
/* The first few digits are special cases.          Firstly, there must be
   at least one digit.          Secondly, the second "digit" may be 'X' or 'x'
   if the radix is hexadecimal.          */
stl_radixSet:
asm          cmp     al, '0'
asm          jne     stl_digitNeeded
asm          cmp     cl, 16          /* is "0x.." allowed ?          */
asm          jne     stl_nextWordDigitJmp
asm          dec     W0 (width)
asm          jl      stl_resultJmp          /* DI:SI is the result          */
          ct ++;
          Get (srceP);
asm          cmp     al, 'x'
asm          je      stl_nextWordDigitJmp
asm          cmp     al, 'X'          /* continue main part of the number */
asm          je      stl_nextWordDigitJmp
asm          jmp     stl_inspectDigit
stl_EOF:     /* source ended before any digit seen          */
          status = EOF;
asm          jmp     short stl_backUp
/* When a syntax error occurs, the result is always zero.          */
stl_badRadix:
stl_noDigitSeen:
          status = 0;
stl_backUp:
          UnGet ( _AX, srceP);
          ct --;
asm          sub     ax, ax
asm          cwd
          goto stl_end;
stl_resultJmp:
asm          jmp     stl_result          /* extend jump range          */
/* Automatic radix recognition:
   Note: single digit "0" with no following digits is a valid octal zero. */
stl_autoRadix:
asm          cmp     al, '0'
          radix = 10;          /* if first digit not '0', numeral is decimal. */
asm          jne     stl_digitNeeded
asm          dec     W0 (width)
asm          jl      stl_resultJmp
          ct ++;
          Get (srceP);
          radix = 8;          /* if begins "0.." then will be octal          */
asm          cmp     al, 'x'          /* unless "0x.." or "0X..", which is hex */
asm          je      stl_autoHex
asm          cmp     al, 'X'
asm          jne     stl_inspectDigit
stl_autoHex:
          radix = 16;
stl_nextWordDigitJmp:
asm          jmp     short stl_nextWordDigit
/* If arrived here then a first true digit is still awaited.          */
stl_digitNeeded:
asm          mov     cx, radix
asm          xchg    bx, ax
asm          call    stl_Digit
asm          xchg    ax, bx
asm          jc      stl_noDigitSeen
asm          xchg    si, ax
asm          jmp     short stl_nextWordDigit
stl_digitOnWord:
asm          xchg    ax, si
asm          mul     W0 (radix)
asm          add     si, ax

```

```

asm      adc      di, dx
asm      jnz      stl_nextDigit
stl_nextWordDigit:
asm      dec      W0 (width)
asm      jl       stl_result
      ct ++;
      Get (srceP);
stl_inspectDigit:
asm      mov      cx, radix
asm      xchg     bx, ax
asm      call     stl_Digit
asm      xchg     ax, bx
asm      jnc      stl_digitOnWord
asm      jmp      short stl_term
/* Loop accumulating digits until overflow or the end of the digits.
   The loop calculation is DI:SI = (DI:SI * radix) + new digit. */
stl_digitOnLong:
asm      xchg     ax, si
asm      mul      cx          /* CX == radix */
asm      xchg     ax, di
asm      xchg     cx, dx
asm      mul      dx
asm      add      si, di
asm      adc      ax, cx
asm      xchg     di, ax      /* result in DI:SI */
asm      adc      dl, dh      /* bits beyond 32nd should be zero. */
asm      jnz      stl_overflow
stl_nextDigit:
asm      dec      W0 (width)
asm      jl       stl_result
      ct ++;
      Get (srceP);
asm      mov      cx, radix
asm      xchg     bx, ax
asm      call     stl_Digit
asm      xchg     ax, bx
asm      jnc      stl_digitOnLong
/* Arrive here if no error but terminator character has been seen.
   The original terminator character must be in AL. */
stl_term:
      UnGet (_AX, srceP);      /* unget to the terminator. */
      ct --;
stl_result:
asm      mov      dx, di
asm      xchg     ax, si      /* result is now in DX:AX */
/* Was '2' negative seen? If so, then negate the result. */
if (sign)
{
asm      neg      dx
asm      neg      ax
asm      stb      dx, 0      /* negate (DX:AX) */
}
stl_end:
asm      LES      di, countP
asm      mov      bx, ct
asm      *add     ES [di], bx
asm      LES      di, statusP
asm      mov      bx, status
asm      mov      ES [di], bx
asm      pop      ES
      return;
/* An overflow produces a maximum result. */
stl_overflow:
asm      mov      ax, 0FFFFh
asm      mov      dx, 7FFFh
asm      add      al, sign
asm      adc      ah, 0
asm      adc      dx, 0      /* result 8000:0000h if signed. */
      status = 2;

```



```
asm          jmp      short  stl_end
}
#pragma warn _rvl
#pragma warn _use
```

函数名 __screenio - 移动屏幕数据块。 -- screen.c

用 法 void pascal __screenio(void far *dst, void far *src, int len);

返回值 无。

源程序

```
void pascal __screenio(void far *dst, void far *src, int len)
{
    if (!video.graphicsmode && directvideo)
        vram(dst, src, len);
    else
        bios(dst, src, len);
}
```

函数名 screenpos - 返回当前 CP。 -- screen.c

用 法 static unsigned near pascal screenpos(void far *ptr);

返回值 高/低字节中包含当前行/列。

源程序

```
#include < video.h>
#include < dos.h>
#include < conio.h>
static unsigned near pascal screenpos(void far *ptr)
{
    register unsigned offset;
    unsigned char row, col;
    offset = FP_OFF(ptr) >> 1;
    row = (unsigned char)(offset / _video.screenwidth);
    col = (unsigned char)(offset - row * _video.screenwidth);
    _AH = row;
    _AL = col;
    return _AX;
}
```

函数名 __scroll - 在正文方式下滚动当前窗口。 -- scroll.c

用 法 static pascal __scroll(byte dir, byte x1, byte y1,
 byte x2, byte y2, byte lines);

返回值 无。

源程序

```
void pascal __scroll(byte dir, byte x1, byte y1, byte x2, byte y2, byte lines)
{
    unsigned linebuffer[80];
    if (!video.graphicsmode && directvideo && lines == 1)
    {
        x1++;
        y1++;
        x2++;
        y2++;
        if (dir == UP)
        {
            movetext(x1,y1+1,x2,y2,x1,y1);
            gettext(x1,y2,x1,y2,linebuffer);
            zapline(linebuffer,x1,x2);
            puttext(x1,y2,x2,y2,linebuffer);
        }
        else
        {
            movetext(x1,y1,x2,y2-1,x1,y1+1);
            gettext(x1,y1,x1,y1,linebuffer);
            zapline(linebuffer,x1,x2);
            puttext(x1,y1,x2,y1,linebuffer);
        }
    }
}
```

```

    }
    else
    {
        BH = video.attribute;
        AH = dir;
        AL = lines;
        CH = y1;
        CL = x1;
        DH = y2;
        DL = x2;
        _VideoInt();
    }
}

```

函数名 __searchpath - 返回指定文件的完整 DOS 路径。-- searchp.cas

用 法 char *pascal __searchpath(const char *pathP, int mode);

原型文件 dir.h

说 明 __searchpath 根据指定的文件名定位一个文件。如果 mode 的指定值为_USEPATH，则搜索 MS-DOS 路径。如果文件没找到，并且指定了_USEPATH，则取得 PATH 环境变量，并依次搜索路径的每一个子目录，直到找到文件或路径搜索完。

找到文件时，返回包含完整路径名的字符串。该串可以用于调用 open 或 exec 等存取该文件。返回的字符串存放于一静态缓冲区，以后每次调用 __searchpath 都将改变其内容。

返 回 值 成功：指向该文件名字符串的指针；

失败：NULL。

源 程 序

```

char * pascal __searchpath(const char *pathP, int mode)
{
    register char *bufP = PathFile;
    register char *envP = NULL;
    int flag;

    /* Preliminary checking */
    flag = 0;
    if ((pathP != NULL) || (*pathP != 0))
        flag = fnsplit(pathP, drive, dir, fname, ext);
    if ((flag & (WILDCARDS + FILENAME)) != FILENAME)
        return (NULL);

    /* If looking for a program file, limit the search if a
       directory or an extension is specified */
    if (mode & PROGRAM) {
        if (flag & DIRECTORY)
            mode &= ~USEPATH;
        if (flag & EXTENSION)
            mode &= ~PROGRAM;
    }

    /* Get "PATH" environment variable if allowed */
    if (mode & USEPATH)
        envP = getenv("PATH");

    /* Try to locate "pathP" in current directory, then try in all
       directories specified by the environment variable "PATH" and
       return a pointer to the full path if found, otherwise NULL
       is returned. */
    while (1) {
        /* Check if the file exists */
        if (CheckFile(bufP, drive, dir, fname, ext, mode))
            break;

        /* If PROGRAM file, try with ".COM" and ".EXE" extension */
        if (mode & PROGRAM) {
            if (CheckFile(bufP, drive, dir, fname, ".COM", mode))
                break;
            if (CheckFile(bufP, drive, dir, fname, ".EXE", mode))
                break;
        }

        /* Stops if no environment or end of it */
    }
}

```

```

        if (envP == NULL || *envP == '\0') {
            buffP = NULL;
            break;
        }
        /* Isolate drive name from environment */
        flag = 0;
        if (envP[1] == ':') {
            drive[flag++] = *envP++;
            drive[flag++] = *envP++;
        }
        drive[flag] = 0;
        /* Isolate directory name from environment */
        for (flag = 0; (dir[flag] = *envP++) != 0; flag++)
            if (dir[flag] == '\\') {
                dir[flag] = 0;
                envP++;
                break;
            }
        envP--; /* point back at '\0' or past ':' */
        /* if only drive specified, set dir to root */
        if (dir[0] == '\0') {
            dir[0] = '\\';
            dir[1] = '\0';
        }
    }
    return (buffP);
}

```

searchpath 搜索 DOS 路径

-- searchp.cas

用 法 char *searchpath (char *filename);

原 型 在 dir.h

说 明 searchpath 试图使用 MS-DOS 路径来定位一个由 filename 给定的文件。函数的返回值为一个指向完整路径名字符串的指针。

首先检查当前驱动器下的当前目录，如果没有找到该文件，则取 PATH 环境变量，按顺序搜索路径中的每个目录直到找到该文件或路径搜索完。

当文件定位后，一个包含完整路径名的字符串被返回，该字符串可用于对 open 或 exec... 的调用中以存取文件。

返回的字符串位于一个静态缓冲区中，每次对 searchpath 的后续调用都破坏前一内容。

返 回 值 在成功定位文件后，指向文件名字符串 filename 的指针被返回；否则，searchpath 返回 NULL。

可移植性 只适用于 MS-DOS

参 见 exec..., open, system

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <dir.h>
#include <dir.h>
#include <string.h>
#include <stdlib.h>
char *searchpath(const char *file)
{
    return __searchpath(file, _USEPATH);
}

```

sector 画并填充椭圆扇区

用 法 #include <graphics>

void far sector (int x, int y, int stangle, int endangle,

int xradius, int yradius);

原 型 在 <graphics.h>

说 明 sector 函数以(x,y)为中心, 以 xradius 和 yradius 为长轴, 从起始角(stangle)到终止角(endingangle)并填充扇区。

扇区轮廓线用当前颜色画, 用 setfillstyle 和 setfillpattern 定义的模式和颜色填充. 当 stangle 为 0, endingangle 为 360 时画一完整椭圆, 使用的角方向为逆时针方向。

若在画线或填充扇区时出现错误, graphresult 返回-6。

可移植性 在 Turbo Pascal 5.0 中有相似子程序

参 见 pieslice, setfillstyle, fill_patterns(变量), graphresult

segread 读段寄存器值 -- segread.cas

用 法 # include <dos.h>

void segread (struct SREGS *segtbl);

原 型 在 dos.h

说 明 segregs 把段寄存器的当前值(存在 SEGREGS)放到 segtbl 所指的结构中。

本调用可与 intdosx 和 int86x 一起使用。

返 回 值 无

可移植性 只适用于 MS-DOS

参 见 FP_OFF, intdos, int86

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
void segread(struct SREGS *segp)
{
    #if (LDATA)
    asm    mov     ax, es
    #endif
    asm    LES     si, segp
    #if (LDATA)
    asm    mov     ES_ [si].es, ax
    #else
    asm    mov     ES_ [si].es, es
    #endif
    asm    mov     ES_ [si].cs, cs
    asm    mov     ES_ [si].ss, ss
    asm    mov     ES_ [si].ds, ds
    #if (LDATA)
    asm    mov     es, ax          /* preserve ES */
    #endif
}
```

setactivepage 设置图形输出活动页

用 法 # include <graphics.h>

void far setactivepage (int pagenum);

相关函数

用 法 void far setvisualpage (int pagenum);

原 型 在 graphics.h

说 明 setactivepage 使 pagenum 成为活动的图形页, 其后所有的图形输出都针对 pagenum 图形页。setvisualpage 使得 pagenum 成为可见图形页。

活动图形页可以是也可以不是在屏幕上看到的页, 这取决于系统有多少有效图形页。只有

EGA,VGA 和 Hercules 图形卡支持多个图形页。

有了多个图形页,程序就可以将图形输出到一个并闭屏幕页,然后通过调用 setvisualpage 改变可见页来快速显示关闭屏幕图象,该技术在动画中特别有用。

返回值 无

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

setallpalette 按指定方式改变所有调色板颜色

用法 #include <graphics.h>

void far setallpalette (struct palette far *palette);

原型在 graphics.h

说明 见 getpalette

setaspectratio 设置图形纵横比

用法 #include <graphics.h>

void far setaspectratio (int xasp, int yasp);

原型在 graphics.h

说明 本函数用于改变当前图形模式的纵横比。以便使 arc 或其它相似函数画出的弧是圆的而不是扁的。通常 yasp 应为 10000, 如想使象点呈正方形(VGA 方式), xasp 应为 10000; 如想使象素点高大于宽, xasp 应小于 10000。可用 getaspectratio 读取当前图形的纵横比。

参见 getaspectratio

setbkcolor 设置当前背景颜色

用法 #include <graphics.h>

void far setbkcolor(int color);

原型在 graphics.h

说明 见 getbkcolor

setblock 修改先前已分配的 DOS 存储段大小 -- setblock.cas

用法 int setblock (int seg, int newsize);

原型在 dos.h

说明 见 allocmem

源程序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
#include <io.h>
int setblock(unsigned segx, unsigned newsize)
{
    asm    mov     ah, 4ah
    asm    mov     bx, newsize
    asm    mov     es, segx
    asm    int     21h
    asm    jc      setblockFailed
    return(-1);
setblockFailed:
    asm    push    bx
    asm    _IOerror(AX);
    asm    pop     ax
    return (AX);
}
```

setbuf 把缓冲区与流相联

-- setbuf.c

用 法 #include <stdio.h>

void setbuf(FILE *stream, char *buf);

相关函数

用 法 int setvbuf(FILE *stream, char *buf, int type, unsigned size);

原 型 在 stdio.h

说 明 setbuf 和 setvbuf 使得 I/O 缓冲区用 buf 缓冲区而不是自动分配，它们在流 stream 打开后使用。

在 setbuf 中，如果 buf 为 NULL，I/O 不缓冲；否则，将全部缓冲。缓冲区的长度必须为 BUFSIZE 字节长(在 stdio.h 中指明)。在 setvbuf 中，如果 buf 为 NULL，将使用 malloc 分配缓冲区。参数 size 指明缓冲区大小，它必须大于零。

如果没有重定向，stdin 和 stdout 将不缓冲；否则，它们将全部缓冲。setbuf 可用于改变使用的缓冲格式。

不缓冲意味着写到流中的字符将立即输出到文件或设备中；缓冲意指字符被累积起来作为块来写。

在 setvbuf 中，参数 type 为下列值之一：

 _IOFBF 文件全部缓冲。当缓冲区为空时，下一输入操作试图填满整个缓冲区。对于输出，在写数据到文件前，先填满整个缓冲区。

 _IOLBF 文件行缓冲。当缓冲区空时，下一输入操作试图填满整个缓冲区。对于输出，当换行字符写到文件上时，缓冲区被清除。

 _IONBF 文件不缓冲，参数 buf 和 size 被忽略，每一输入操作直接从文件读，每一输出操作直接写数据到文件中。

除非在打开流 stream 或调用了 fseek 之后，setbuf 用于 stream 将产生不可预测的结果。在流打开而不缓冲情况下调用 setbuf 是合法的，不会引起任何问题。

出现错误的通常原因是把缓冲区当作自动(局部)变量分配，而从定义缓冲区的函数中返回时，不能关闭文件。

返 回 值 setbuf 不返回值

setvbuf 成功时返回 0。如果给定 type 和 size 为无效值，buf 为 NULL 或无足够空间分配一缓冲区，将返回非零值。

可移植性 适用于 UNIX 系统

参 见 fopen, fclose, fseek, malloc, open

源 程 序

```
#include <stdio.h>
void setbuf(FILE *fp, char *buf)
{
    setvbuf (fp, buf, (buf != NULL) ? _IOFBF : _IONBF, BUFSIZ);
}
```

setcbreak 取得 control-break 设置

-- getcbreak.c

用 法 int setcbreak (int value);

原 型 在 dos.h

说 明 见 getcbreak

源 程 序

```
#include <dos.h>
int setcbreak(int eval)
{
    _AX = 0x3301;
```

```

        DL = cval;
        geninterrupt(0x21);
        return(_DL);
    }

```

setcolor 设置当前画线颜色

用 法 #include <graphics.h>

```
void far setcolor (int color);
```

原型在 graphics.h

说 明 见 setbkcolor

setdate 设置 MS-DOS 日期

-- setdate.cas

用 法 #include <dos.h>

```
void setdate (struct date *dateblk);
```

原型在 dos.h

说 明 见 getdate

源 程 序

```

#pragma inline
#include <asmrules.h> #include <dos.h>
void setdate(struct date *dateblk)
{
    asm      mov     ah,2bh
    asm      LES     si, dateblk
    asm      mov     cx, W0(ES [si])
    asm      mov     dx, W1(ES [si])
    asm      int     21h
}

```

setdisk 设置当前磁盘驱动器

-- chdir.cas

用 法 int setdisk (int drive);

原型在 dir.h

说 明 见 getdisk

源 程 序

```

int setdisk(int drive)
{
    asm      mov     ah, 00Eh
    asm      mov     dl, drive
    asm      int     021h
    asm      xor     ah,ah
    return _AX;
}

```

setdta 设置磁盘传输地址

-- setdta.cas

用 法 void setdta (char far *dta);

原型在 dos.h

说 明 见 getdta

源 程 序

```

#pragma inline
#include <dos.h>
void setdta(char far *dta)
{
    asm      push     ds
    asm      mov     ah, 01Ah

```

```
asm    lds     dx, dta
asm    int     021H
asm    pop     ds
```

setfillpattern 选择用户定义的填充模式

用 法 # include <graphics.h>

```
void far setfillpattern(char far *upattern, int color);
```

原型在 graphics.h

说 明 见 getfillpattern

setfillstyle 设置填充模式和颜色

用 法 # include <graphics.h>

```
void far setfillstyle (int pattern, int color);
```

原型在 graphics.h

说 明 见 getfillstyle

settime 取得文件日期和时间

-- settime.cas

用 法 # include <io.h>

```
int settime (int handle, struct ttime *ftimep);
```

原型在 io.h

说 明 见 gettime

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <io.h>
#include <ip.h>
int settime(int handle, struct ttime *ftimep)
{
asm    mov     ax, 05701h
asm    mov     bx, handle
asm:    LES     di, ftimep
asm    mov     cx, ES [di]
asm    mov     dx, ES [di+2]
asm    int     021h
asm    jc      settimeFailed
asm    return (0);
settimeFailed:
    return  _IOerror (_AX);
}
```

setgraphbufsize 改变内部图形缓冲区的大小

用 法 # include <graphics.h>

```
unsigned far setgraphbufsize (unsigned bufsize);
```

原型在 graphics.h

说 明 一些图形例行程序(如 floodfill)要使用调用 initgraph 时所分配的存储缓冲区, 该缓冲区在调用 closegraph 时被释放。由 _graphgetmem 所分配的这个缓冲区的缺省大小为 4096 字节。可使缓冲区小一些(节省存储空间)或使它大一些(如调用 floodfill 时产生错误-7: Out of flood memory)。setgraphbufsize 通知 initgraph 调用 _graphgetmem 时为内部图形缓冲区分配多少存储。调用 setgraphbufsize 必须在调用 initgraph 之前。

返回值 setgraphbufsize 返回内部缓冲区的原来大小。

可移植性 在 Turbo Pascal 5.0 中有相似的程序。

参见 closegraph, initgraph

setgraphmode 将系统设置成图形模式且清屏

用法 #include <graphics.h>

void far setgraphmode (int mode);

原型在 graphics.h

说明 见 getgraphmode

setjmp 非局部转移

-- setjmp.cas

用法 #include <setjmp.h>

int setjmp(jmp_buf env);

原型在 setjmp.h

说明 见 longjmp

源程序 #pragma inline

#include <asmrules.h>

#include <setjmp.h>

int setjmp(jmp_buf jmpb)

{

#if !(LDATA)

 _ES = _DS;

#endif

asm mov dx, es /* preserve es, di */

asm mov cx, di

asm LES di, jmpb

asm cld

asm lea ax, jmpb

asm stosw /* sp */

asm mov ax, SS

asm stosw /* SS */

asm pushf

asm pop ax

asm stosw /* FL */

#if (LPROC)

asm mov ax, W1(jmpb-cPtrSize) /* large code */

#else

asm mov ax, CS

#endif

asm stosw /* CS */

asm mov ax, W0(jmpb-cPtrSize)

asm stosw /* IP */

asm mov ax, [bp]

asm stosw /* BP */

asm mov ax, cx

asm stosw /* DI */

asm mov ax, dx

asm stosw /* ES */

asm mov ax, si

asm stosw /* SI */

#if defined(__HUGE__)

asm pop ax

asm push ax /* caller's DS */

#else

asm mov ax, ds

#endif

asm stosw /* DS */

asm mov es, dx /* restore ES */

return(0);

}

setlinestyle 设置当前画线宽度和类型

用 法 #include <graphics.h>

```
void far setlinestyle (int linestyle, unsigned upattern,  
int thickness);
```

原 型 在 graphics.h

说 明 见 getlinesettings

setmem 存值到存储区 -- memset.cas

用 法 void setmem (void *addr, int len, char value);

原 型 在 mem.h

说 明 见 movemem

源 程 序

```
#pragma inline  
#include <asmrules.h>  
#include <mem.h>  
void setmem(void *addr, unsigned len, char val)  
{  
#if !!(LlDdTA)  
    _ES = _DS;  
#endif  
asm LES di, addr  
asm mov cx, len  
asm mov al, val  
asm mov ah, al  
asm cld  
asm test di, 1  
asm jz isAligned  
asm jcxz done  
asm stosb  
asm dec cx  
isAligned:  
asm shr cx, 1  
asm rep stosw  
asm jnc noOdd  
asm stosb  
noOdd:  
done: ;  
}
```

setmode 设置打开文件方式 -- setmode.c

中 法 int setmode (int handle, unsigned mode);

原 型 在 io.h

说 明 setmode 设置与 handle 相联的打开文件为二进制或文本方式。参数 mode 必须为 O_BINARY 和 O_TEXT 之一，或都不给出。

返 回 值 成功时，setmode 返回 0；出错时，返回-1 并置 errno 为：

EINVAL 无效参数

可移植性 适用于 UNIX 系统

参 见 fread, read, fmode(变量)

源 程 序

```
#include <io.h>  
#include <asmrules.h>  
#include <io.h>  
#include <fcntl.h>
```

```

int setmode(int fildes, register int mode)
{
    register int newmode;
    if (fildes < 0 || fildes >= HANDLE_MAX || _openfd[fildes] == ~0U)
        return (__IOerror(e_badHandle));
    if ((newmode = mode & (O_TEXT | O_BINARY)) == mode &&
        newmode != (int)(O_TEXT | O_BINARY))
    {
        mode = _openfd[fildes];
        _openfd[fildes] = (mode & ~(O_TEXT | O_BINARY)) | newmode;
        return (mode & (O_TEXT | O_BINARY));
    }
    else
        return (__IOerror(e_badFunction));
}

```

setpalette 设置调色板入口

用 法 #include <graphics.h>

void far setpalette(int index, int actual_color);

原 型 在 graphics.h

说 明 getallpalette

setrgbcolor 设置 VGA 或 IBM-8514 驱动程序的调色板入口

用 法 #include <graphics.h>

void far setrgbcolor (int colornum, int red, int green, int blue);

原 型 在 graphics.h

说 明 setrgbcolor 用于设置 VGA 或 IBM-8514 图形驱动程序的调色板入口, 其中 colornum 是待设置入口名, red, green 和 blue 是它的几个颜色分量。

对于 IBM-8514 显示卡, colornum 的范围是 0-255, 在 256 种颜色模式的 VGA 卡中, colornum 的范围为 0-15。

参 见 setpalette, setrgbpalette

setrgbpalette 定义 IBM-8514 图形卡的颜色

用 法 #include <graphics.h>

void far setrgbpalette(int colornum, int red,
int green, int blue);

原 型 在 graphics.h

说 明 colornum 是待加载的调色板入口, 范围为 0-255, red, green 和 blue 定义分量颜色。这些值中, 只有低字节的 6 个有效位被装入调色板

参 见 setrgbcolor, setpalette

setswitchchar 设置 MS-DOS 开关字符值。 -- getswit.c

用 法 void setswitch(char byte);

原型文件 dos.h

说 明 将当前 MS-DOS 开关字符值设置为 byte 的值。该调用使用了未公开的 MS-DOS 系统调用 0X3F, 并在 MS-DOS 2.0 到 3.0 下工作 (以后的 MS-DOS 版本可能改变或不支持该调用)。

返 回 值 无。

源 程 序

```

void setswitch(char ch)
{
    _AX = 0x3701;
}

```

```

        DL = ch;
        _geninterrupt(0x21);
    }

```

settextjustify 设置文本的对齐方式

用 法 #include <graphics.h> void far settextjustify (int horiz, int vert);
 原型在 graphics.h
 说 明 见 gettextsettings

settextstyle 设置当前文本特性

用 法 #include <graphics.h>
 void far settextstyle (int font, int direction, int charsize);
 原型在 graphics.h
 说 明 见 gettextsettings

settime 设置系统时间

-- setdate.cas

用 法 #include <dos.h>
 void settime (struct time *timep);
 原型在 dos.h
 说 明 见 gettime
 源 程 序

```

#pragma inline
#include <asmrules.h>
#include <dos.h>
void setttime(struct time *timeblk)
{
    asm      mov     ah, 2dh
    asm      LES     si, timeblk
    asm      mov     cx, W0(ES_ [si])
    asm      mov     dx, W1(ES_ [si])
    asm      int     21h
}

```

setusercharsize 用于矢量字体的用户定义字符放大因子

用 法 #include <graphics.h>
 void far setusercharsize(int multx, int dirx,
 int multy, int diry);

原型在 graphics.h

说 明 setusercharsize 提供很好地控制矢量字体文本大小的手段。只有当先前调用 settextstyle 时置 charsize=0, 由 setusercharsize 设置的值才是有效的。

使用 setusercharsize, 用户可以指定宽度和高度的比例因子, 缺省的宽度由 multx 给定, 高度由 multy 给定。如想使文本宽度二倍于缺省值, 高度比缺省值高 50%, 可置:

```

    multx = 2; dirx = 1;
    multy = 3; diry = 2;

```

返回值 无

可移植性 在 Turbo Pascal 5.0 中有相似的子程序

参 见 gettextsettings

setvbuf 把缓冲区与流相联

-- setvbuf.c

用 法 #include <stdio.h>

int setvbuf (FILE *stream, char *buf, int type, unsigned size);

原 型 在 stdio.h

说 明 见 setbuf

源 程 序

```
#include <stdio.h>
#include <stdlib.h>
extern void (*_exitbuf)();
extern void _xflush();
int _stdinStarted = 0;
int _stdoutStarted = 0;
int setvbuf(register FILE *fp, char *buf, int type, size_t size)
{
    if (fp->token != (short) fp || _IONBF < type || 0x7fff < size)
        return (EOF);
    if (! _stdoutStarted && ((short) fp == (short) stdout))
        _stdoutStarted = 1;
    else
        if (! _stdinStarted && ((short) fp == (short) stdin))
            _stdinStarted = 1;
    /* Ensure the change in buffering causes no loss of characters.
       fseek() will flush and reposition safely. */
    if (fp->level)
        fseek (fp, 0L, SEEK_CUR);
    if (fp->flags & _F_BUF)
        free (fp->buffer);
    fp->flags &= ~(_F_BUF | _F_LBUF);
    fp->bsize = 0;
    fp->curp = fp->buffer = & fp->hold;
    if (_IONBF != type && size > 0)
    {
        _exitbuf = _xflush;
        if (NULL == buf)
        {
            if ((buf = malloc (size)) != NULL)
                fp->flags |= _F_BUF;
            else
                return (EOF);
        }
    }
    #pragma warn -ucp
    fp->buffer = fp->curp = buf;
    #pragma warn .ucp
    fp->bsize = size;
    if (_IOLBF == type)
        fp->flags |= _F_LBUF;
    return (0);
}
```

setvect 设置中断矢量入口

-- getvect.cas

用 法 void setvect (int intr_num, void interrupt (*isr)());

原 型 在 dos.h 说 明 见 getvect

源 程 序

```
#pragma inline
#include <dos.h>
void setvect(int intr, void interrupt (far* func)())
{
    asm      mov     ah, 025h
    asm      mov     al, intr
    asm      push    ds
    asm      lds     dx, dword ptr func
    asm      int     021h
    asm      pop     ds
}
```

setverify 设置验证状态 -- getverf.cas

用 法 void setverify (int value);

原 型 在 dos.h

说 明 见 getverify

源 程 序

```
#pragma inline
#include <dos.h>
void setverify(int value)
{
asm    mov    ah, 02Eh
asm    mov    al, value
asm    int    021h
}
```

setviewport 为图形输出设置当前视区

用 法 # include <graphics.h>

```
void far setviewport(int left, int top, int right,
                    int bottom, int clipflag);
```

原 型 在 graphics.h

说 明 见 getviewport

setvisualpage 设置可见图形页号

用 法 # include <graphics.h>

```
void far setvisualpage (int pagenum);
```

原 型 在 graphics.h

说 明 见 setactivepage

setwritemode 设置画线的输出模式

用 法 # include <graphics.h>

```
void far setwritemode (int mode);
```

原 型 在 graphics.h

说 明 setwritemode 函数用于设置画线的输出模式。若模式为 0, 新画的线将覆盖屏幕上原有的图象; 若模式为 1, 新线的象素点与旧线象素点之间先进行“异或(XOR)”, 后往屏幕输出。

可移植性 在 Turbo Pascal 5.0 中有相似的程序

参 见 lineto

signal 设置某一信号的对应动作 -- signal.c

用 法 # include <signal.h>

```
int signal (int sig, sigfun fname);
```

原 型 在 signal.h

说 明 signal 函数用于指定某一信号的对应动作, 该信号是由 raise 函数激为或发生异常情况时产生的, sigfun 是一指向函数的指针类型:

```
typedef void (*sigfun)(int subcode);
```

fname 为一动作函数, 已定义的动作函数有:

SIG_DFL 终止程序执行(缺省)

SIG_IGN 忽略该信号
 SIG_ERR 返回错误代码

用户可以定义自己的动作函数

参 见 raise

源 程 序

```
void (*_Cdecl signal(sig, New))(int)
int sig;
CatcherPTR New;
/* CatcherPTR signal(register int sig, CatcherPTR New) */
{
    register int Index;
    CatcherPTR OldVal;
    if (!SignalPtrSet) /* let _fperr() know where signal is */
    {
        SignalPtr = signal;
        SignalPtrSet = 1;
    }
    /* For OS/2 some default settings will be 'ignore's, look'em up */
#ifdef __OS2__
#define DISPATCH_SETTING ((New == SIG_DFL) ? Defaults[Index] : New)
#else
#define DISPATCH_SETTING New
#endif
    /* -----
       Get an index for the signal type, if its had exit.
       ----- */
    if ((Index = GetIndex(sig)) == BogusSignal)
    {
        errno = EINVAL; /* Bogus 'sig' parm was passed */
        return SIG_ERR;
    }
    /* -----
       Install handler (SIGINT, SIGFPE & SIGSEGV are special)
       ----- */
    OldVal = Dispatch[Index]; /* Save the OLD handler */
    Dispatch[Index] = DISPATCH_SETTING; /* Set the NEW handler */
    if (sig == SIGINT)
    {
#ifdef __OS2__
        DOSSETSIGHANDLER(
            (DFAR) 0L,
            (WFAR) 0L,
            (WORD) 2, /* Install handler */
            (WORD) 1 /* SIGINTR */
        );
#else
        setvect(0x23, Int23Catcher); /* Take INT 23H */
#endif
    }
    else if (sig == SIGFPE)
    {
        setvect(0, Int0Catcher); /* Take INT 0 */
        setvect(4, Int4Catcher); /* Take INT 4 */
    }
#ifdef BOUNDS_TRAP
    else if (sig == SIGSEGV)
    {
        if (!GotInt5)
        {
#ifdef __OS2__
            BiosPtrScr = getvect(5); /* Save old INT 5 */
#else
            setvect(5, Int5Catcher); /* Take INT 5 */
            GotInt5 = 1;
#endif
        }
    }
#endif
#ifdef ILLEGALOP_TRAP
    else if (sig == SIGILL)
    {
        setvect(6, Int6Catcher); /* Take INT 6 */
    }
}
```

```
#endif
    return OldVal;
}
```

sin 三角正弦函数

-- sin.cas

用法 double sin (double x);

原型在 math.h

说明 见 trig

源程序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#include <errno.h>
#include <stddef.h>
static unsigned short NANTRIG [4] = {0,0,0x0420, 0x7FF0};
#pragma warn -rvl
double sin (double x) {
asm    FLD    DOUBLE (x)
asm    mov    ax, 7FF0h
asm    and    ax, W0 (x [6])    /* extract the exponent field */
asm    cmp    ax, (53 * 16) + 3FF0h /* biased version of exponent 53 */
asm    jae    sin tooLarge
asm        if (_8087 >= 3)
{
asm    db    OPCODE_FSIN.
}
else
{
asm    FAST_ (_FSIN_)
}
sin_end:
    return;
sin tooLarge:
asm    FSTP    ST (0)    /* pop x from stack */
#pragma warn -ret
    return matherr (TLOSS, "sin", &x, NULL, *((double *) NANTRIG));
#pragma warn .ret
}
#pragma warn .rvl
```

sinh 双曲正弦函数

-- sinh.cas

用法 double sinh (double x);

原型在 math.h

说明 见 hyperb

源程序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <_math.h>
#include <errno.h>
#include <stddef.h>
#pragma warn -rvl
double sinh (double x)
{
asm    FLDI
asm    mov    ax, 7FFFh
asm    FCHS    /* TOS = -1.0 */
asm    mov    dx, x [6]
asm    and    ax, dx /* select exponent and most signif. bits */
asm    FLD    DOUBLE (x)
```



```

asm    cmp    ax, 4086h
asm    jnb    sinh_tooBig    /* exp (+710.475) is the limit */
asm    FABS   asm    cmp    ax, 3FD6h
asm    jb     sinh_small
sinh_justFits:
asm    FAST_  (_FEXP_)
asm    FLD1
asm    FDIV   st, st(1)    /* Exp (-x) */
asm    FSUBP  st(1), st
asm    FSCALE    /* sinh (x) = (exp(x) - exp(-x)) / 2 */
sinh_end:
asm    FSTP   st(1)    /* discard the -1 */
/* change sign if original argument was negative */
asm    test   dh, 80h
asm    jz     sinh_end2
asm    FCHS
sinh_end2:
    return;
sinh_tooBig:
asm    ja     sinh_over
asm    cmp    W0 [x [4]], 33CEh
asm    jb     sinh_justFits
sinh_over:
asm    FCOMPP    /* discard ST and ST(1) */
#pragma warn -ret
    return _matherr (OVERFLOW, "sinh", &x, NULL,
        (_DX & 0x8000) ? - HUGE_VAL : HUGE_VAL);

#pragma warn .ret
sinh_small:
asm    cmp    ax, 3DE0h
asm    jb     sinh_end    /* x tiny, return x */
asm    FLDL2E
asm    FMUL
asm    F2XM1
/* TOS = y = exp(x) - 1 */
asm    FLD1
asm    FADD   st(0),st(1)
/* stack = 1+y,y,-1 */
asm    FDIVR  st(0),st(1)
asm    FADD
asm    FSCALE
    goto sinh_end;
}
#pragma warn .rvl

```

sleep 执行挂起一段时间 -- sleep.c

用 法 unsigned sleep(unsigned seconds);

原 型 在 dos.h

说 明 调用 sleep 时, 当前程序暂停执行, 挂起由参数 seconds 指示的秒数。间断时间精确到百分之一秒, 或 MS-DOS 时钟(不很精确)。

返 回 值 无

可移植性 适用于 UNIX 系统

源 程 序

```

#include <dos.h>
void sleep(unsigned seconds)
{
    struct time    t;
    register int    secs;
    register int    hunds;
    gettime(&t);
    hunds = (t.ti_hund > 90) ? 90 : t.ti_hund;
    while (seconds--)
    {
        secs = t.ti_sec;
        do

```

```

        gettime(&t);
        while (secs == t.ti_sec);
    }
    do
        gettime(&t);
        while (hunds > t.ti_hund);
}

```

sopen 打开一共享文件

用 法 `#include <fcntl.h>`
`#include <sys/stat.h>`
`#include <share.h>`
`#include <io.h>`
`int sopen (char *pathueme, int access, int shflag, int permiss);`

原 型 在 `io.h`

说 明 见 `open`

sound 以指定频率打开 PC 扬声器

-- `sound.cas`

用 法 `void sound (unsigned frequency);`

相关函数

用 法 `void nosound (void);`

原 型 在 `dos.h`

说 明 调用 `sound` 可以按给定的频率打开 PC 扬声器。`frequency` 为以赫兹为单位的频率。如果调用 `sound` 之后又想关闭扬声器,可以调用 `nosound` 函数。

返 回 值 无

可移植性 只适用于 IBM PC 及其兼容机,在 Turbo Pascal 中有相似的程序。

参 见 `delay`, `sleep`

源 程 序

```

#pragma inline
#include <dos.h>
void sound(unsigned frequency)
{
    mov     bx, frequency
    mov     ax, 34DDh
    mov     dx, 0012h
    cld
    jmp     stop
    div     bx
    mov     bx, ax
    in      al, 61h
    test    al, 3
    jne     j1
    or      al, 3
    out     61h, al
    mov     al, 0B6h
    out     43h, al
j1:
    mov     al, bl
    out     42h, al
    mov     al, bh
    out     42h, al
stop:
}

```

spawn... 创建并运行子进程

-- `spawnl.c`

```
-- spawnle.c
-- spawnlp.c
-- spawnlpe.c
-- spawnv.c
-- spawnve.c
-- spawnvp.c
-- spawnvpe.c
```

用 法 #include <process.h>

```
int spawnl(int mode, char *pathname, char *arg0, arg1, ...,
            argn, NULL);
int spawnle(int mode, char *pathneme, char *arg0, arg1, ...,
            argn, NULL, char *envp[]);
int spawnlp(int mode, char *pathname, char *arg0, arg1, ...,
            argn, NULL);
int spawnlpe(int mode, char *pathname, char *arg0, arg1, ...,
            argn, NULL, char *envp[]);
int spawnv (int mode, char *pathname, char *argv[]);
int spawnve (int mode, char *pathname, char *argv[], char *envp[]);
int spawnvp (int mode, char *pathneme, char *argv[]);
int spawnvpe (int mode, char *pathname, char *argv[], char *envp[]);
```

原 型 在 process.h

说 明 spawn 系列函数创建并运行称为子进程的其它文件。必须有足够的存储区用来加载执行这些子进程。

mode 值确定调用函数(父进程),在调用 spawn 后所采取的动作:

P_WAIT 父进程“挂起”直到子进程执行完成;
P_NOWAIT 父进程和子进程同时运行;
P_OVERLAY 子进程覆盖父进程原来的存储区位置,同exec...调用相同。

pathname 是被调用子进程的文件名。spawn...函数调用标准的 MS-DOS 搜索算法查找 pathname:

- * 如果没有扩展名或句点: 先找该名, 若没找到, 加上.COM(或.EXE)扩展名再找;
- * 给定扩展名: 查找该出的文件;
- * 给出句点: 查找无扩展的名。

加在 spawn...系列函数后的后缀 l,v,p,e 表示命名函数的某种操作能力:

- p 表明函数还将在DOS的PATH环境变量所指明的目录中查找子文件。如果没有p后缀,只在根目录和当前工作目录中查找。
- l 表明指针参数arg0, arg1, ..., argn作为独立参数传送。通常, l后缀用于事先知道待传送的参数个数情况。
- v 表明指针参数arg0, arg1, ..., argn作为指针数组传送。通常, v后缀用于要传送参数个数可变的情况。
- e 表明参数envp可以传到子进程, 用于改变了子进程的环境。在没有e后缀情况下, 子进程继承父进程的环境

spawn...系列函数中的每一个都必须有一个参数说明后缀(l 或 v),但路径搜索和环境继承后缀(p 或 e)是可选择的。

例如:

- * spawn 是 spawn...函数之一, 它使用独立参数, 只在根目录或当前目录下搜索子文件,

父进程环境传到子进程

- * `spawnpe` 是一 `spawn...` 函数; 它接受参数指针数组, 在搜索子进程过程中使用 `PATH`, 并接受 `envp` 环境参数用于改变子程序环境

`spawn...` 函数必须至少传送一个参数给子进程(`arg0` 或 `arg[0]`); 该参数约定为 `pathname` 的一个拷贝。

当使用 `l` 后缀时, `arg0` 通常用于指向 `pathname`, 而 `arg1,...,argn` 指向组成新参数表的字符串。`argn` 后的 `NULL` 表示表结束。

当使用 `e` 后缀时, 通过参数 `envp` 传递一新的环境设置表, 这一环境参数表是一 `char *` 数组, 每一元素指向如下形式的空字符终结的字符串:

`envvar = value`

其中 `envvar` 为环境变量名, `value` 是 `envvar` 设置的值。`envp[0]` 的最后一个元素是 `NULL`。如果 `envp[0]` 为 `NULL`, 表明子进程继承父进程的环境设置。

`arg0+arg1+...argn` (或 `arg[0]+arg[1]+...+arg[n]`) 的组合长度 (包括参数间的空格符) 必须小于 128 字节。空字符终结符不算在内。

当调用 `spawn...` 函数时, 原来打开的文件在子进程中仍然打开。

返回值 在执行成功时, 返回值为子进程退出状态 (0 为正常终结)。如果子进程调用带非零参数 `exit`, 退出状态可以设置为非零值。

在出现错误时, `spawn...` 函数返回 -1, `errno` 被置为下列值之一:

<code>E2BIG</code>	参数表太长
<code>EINVAL</code>	无效参数
<code>ENOENT</code>	路径或文件名没有找到
<code>ENOEXEC</code>	<code>EXEC</code> 格式错
<code>ENOMEM</code>	无足够存储区

参 见 `abort`, `atexit`, `exit`, `exec...`, `system`

源 程 序

```
#include <process.h>
#include <_process.h>
#include <errno.h>
#include <stddef.h>
int spawnl(int modeF, char *pathP, char *arg0, ...)
{
    register int (*Func)();
    switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
        case P_OVERLAY :
            Func = _exec;
            break;
        default :
            errno = EINVAL;
            return (-1);
    }
    return _LoadProg(Func, pathP, &arg0, NULL, 0);
}
int spawnle(int modeF, char *pathP, char *arg0, ...)
{
    register char **p;
    register int (*Func)();
    /* Find the end of the argument list */
    for (p = &arg0; *p++ != NULL; );
    switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
```

```

        case P_OVERLAY :
            Func = _exec;
            break;
        default :
            errno = EINVAL;
            return (-1);
    }
    return _LoadProg(Func, pathP, &arg0, (char **)p, 0);
}

int spawnlp(int modeF, char *pathP, char *arg0, ...)
{
    register int    (*Func)();
    switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
        case P_OVERLAY :
            Func = _exec;
            break;
        default :
            errno = EINVAL;
            return (-1);
    }
    return _LoadProg(Func, pathP, &arg0, NULL, 1);
}

int spawnlpe(int modeF, char *pathP, char *arg0, ...)
{
    register char    **p;
    register int    (*Func)();
    /* Find the end of the argument list */
    for (p = &arg0; *p++ != NULL; );          switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
        case P_OVERLAY :
            Func = _exec;
            break;
        default :
            errno = EINVAL;
            return (-1);
    }
    return _LoadProg(Func, pathP, &arg0, (char **)p, 1);
}

int spawnv(int modeF, char *pathP, char *argv[])
{
    register int    (*Func)();
    switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
        case P_OVERLAY :
            Func = _exec;
            break;
        default :
            errno = EINVAL;
            return (-1);
    }
    return _LoadProg(Func, pathP, argv, NULL, 0);
}

int spawnve(int modeF, char *pathP, char *argv[], char *envV[])
{
    register int    (*Func)();
    switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
        case P_OVERLAY :
            Func = _exec;
            break;
        default :

```

```

        errno = EINVAL;
        return (-1);
    }
    return _LoadProg(Func, pathP, argv, envV, 0);
}
int spawnvp(int modeF, char *pathP, char *argv[])
{
    register int (*Func)();
    switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
        case P_OVERLAY :
            Func = _exec;
            break;
        default :
            errno = EINVAL;
            return (-1);
    }
    return _LoadProg(Func, pathP, argv, NULL, 1);
}
int spawnvpe(int modeF, char *pathP, char *argv[], char *envV[])
{
    register int (*Func)();
    switch(modeF)
    {
        case P_WAIT :
            Func = _spawn;
            break;
        case P_OVERLAY :
            Func = _exec;
            break;
        default :
            errno = EINVAL;
            return (-1);
    }
    return _LoadProg(Func, pathP, argv, envV, 1);
}

```

sprintf 送格式输出到字符串中

sprintf.c

用 法 int sprintf (char *string, char *format [,argument,...]);

原 型 在 stdio.h

说 明 见 printf

源 程 序

```

#include <stdio.h>
#include <mem.h>
#include <string.h>
#include <_printf.h>
int cdecl sprintf(char *bufP, const char *fmt, ...)
{
    *bufP = 0;
    return __vprinter ((putnF *)strputn, &bufP, fmt, _va_ptr);
}

```

sqrt 计算平方根

-- sqrt.cas

用 法 double sqrt (double x);

原 型 在 math.h 说 明 见 exp

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include <math.h>
#include <errno.h>

```

```

#include <stddef.h>
static unsigned short NANSQRT [4] = {0,0,0x0020, 0x7FF0};
#pragma warn -rvl
double sqrt (double x)
{
asm    FLD    DOUBLE (x)
asm    mov    ax, x [6]
asm    shl    ax, 1
asm    jz     sqrt_zero
asm    jc     sqrt_imaginary
asm    FSQRT
sqrt_zero:                /* zero = sqrt (zero) */
sqrt_end:
    return;
sqrt_imaginary:
asm    FSTP    st (0)      /* pop x from stack */
#pragma warn -ret
    return _matherr (DOMAIN, "sqrt", &x, NULL, *((double *) NANSQRT));
#pragma warn .ref
}
#pragma warn .rvl

```

srand 初始化随机数发生器

-- rand.c

用 法 void srand (unsigned seed);

原 型 在 stdlib.h

说 明 见 rand.

源 程 序

```

#include <stdlib.h>
#define MULTIPLIER    0x015a4e35L
#define INCREMENT     1
static long Seed = 1;
void srand(unsigned seed)
{
    Seed = seed;
}

```

sscanf 执行从字符串中的格式化输入

-- sscanf.c 用 法 int sscanf (char *string,

char *format[,argument,...]);

原 型 在 stdio.h

说 明 见 scanf

源 程 序

```

#include <stdarg.h>
#include <stdio.h>
#include <scanf.h>
int cdecl sscanf(const char *buf, const char *fmt, ...)
{
#pragma warn -sus
    return scanner (
        (int (*)(void *)) Get,
        (void (*)(int ch, void *)) UnGet,
        &buf,
        fmt,
        va_ptr
    );
#pragma warn .sus
}

```

stat 读取打开文件信息

-- stat.cas

用 法 # include <sys/stat.h>

int stat (char *pathname, struct stat *buff);

相关函数

用法 `int fstat (char *handle, struct stat *buff);`

原型在 `sys\stat.h`

说明 `stat` 和 `fstat` 把一给定打开文件(可目录)的信息存到 `stat` 结构中。

`stat` 取得由 `pathname` 所指的打开文件或目录信息。

`fstat` 取与句柄 `handle` 相联的打开文件的信息。

在这两个函数中, `buff` 指向 `stat` 结构(在 `sys\stat.h` 中定义), 包括以下几个字段:

<code>st_mode</code>	位屏蔽给出打开文件方式的信息。
<code>st_dev</code>	包含文件或文件句柄(若文件在设备中)的磁盘驱动器号。
<code>st_rdev</code>	同 <code>st_dev</code> 。
<code>st_nlink</code>	置为整型常数1。
<code>st_size</code>	打开文件的字节大小。
<code>st_atime</code>	打开文件最近修改时间。
<code>st_mtime</code>	同 <code>st_atime</code> 。
<code>st_ctime</code>	同 <code>st_atime</code> 。

`stat` 结构还包含三个这里没有提到的字段: 它们包含在 DOS 下无意义的值。

位屏蔽给出打开文件方式的信息, 包括以下各位:

下列一位被设置:

<code>S_IFCHR</code>	如果 <code>handle</code> 指向外设(<code>fstat</code>), 置位。
<code>S_IFREG</code>	如果 <code>handle</code> (<code>fstat</code>) 指向或 <code>pathname</code> (<code>stat</code>) 指向的是普通文件, 置位。
<code>S_IFDIR</code>	如果 <code>pathname</code> 指定一目录(<code>stat</code>), 置位。

以下一位或两位被设置:

<code>S_IWRITE</code>	如果用户有写文件的权限, 置位。
<code>S_IREAD</code>	如果用户有读文件的权限, 置位。

对 `stat`, 位屏蔽还包括用户可执行位; 这些是根据打开文件的扩展设置的。

位屏蔽还包括读/写位; 这些是根据文件的权限方式而设置的。

返回值 在成功检索了打开文件的信息后, 这两个函数均返回 0; 在出错时(无法得到信息), 每一函数都返回 -1, 并置 `error` 值为:

<code>ENOENT</code>	文件或路径没有找到(对 <code>stat</code>)
<code>EBADF</code>	无效文件句柄(对 <code>fstat</code>)

源程序

```
#pragma inline
#include <asmrules.h>
#include <sys\stat.h>
#include <dos.h>
#include <io.h>
int stat (char *pathP, struct stat *bufP)
{   dosSearchInfo   info;
    pushDS
    #if LDATA
asm    push    SS
asm    pop     DS
    #endif
asm    lea     dx, info
asm    mov     ah, 1Ah          /* set Device Transfer Address */
asm    int     21h
    popDS
asm    jc     statFailed
    pushDS
asm    LDS     dx, pathP
asm    mov     cx, 16h          /* include directory, hidden, and system files */
```



```

asm    mov     ah, 4Eh      /* Find First      - */
asm    int     21h
asm    popDS_
asm    jc      statFailed
asm    mov     di, info.attrib
asm    sub     si, si        /* SI = file mode */
/* It is a non-documented feature of the FindFirst function that devices
   can be "found". They are distinguished by the attribute 40h. */
asm    test    di, 40h      /* is it a character stream ? */
asm    jnz     sta_isDevice
/* Arrive here if the info is for a regular file. */
asm    _SI |= S_IREAD;
asm    dec     BY0(info.drive) /* drives from 0..n-1, not 1..n */
asm    test    di, 10h      /* directory ? */
asm    jnz     sta_isDir
asm    _SI |= S_IFREG | S_IREAD;
asm    test    di, 1        /* read only ? */
asm    jnz     sta_convertTime
asm    _SI |= S_IWRITE;      /* write allowed */
asm    jmp     short sta_convertTime
/* Arrive here if MSDOS calls failed. */
statFailed:
    return __IOerror (_AX);
sta_isDir:
    _SI |= S_IFDIR | S_IXEXEC;
/* MSDOS time is a 32-bit record, which must be converted into the Unix
   style of seconds since 1970. */
sta_convertTime:
    _DOStimeToU (*((long *) &info.time));
asm    xchg    cx, ax        /* result in DX:CX */
asm    jmp     short sta_construct
/* Arrive here if FindFirst identified the name as a character device. */
sta_isDevice:
asm    mov     W0 (info.drive), -1
asm    _SI |= S_IFCHR | S_IREAD | S_IWRITE;
asm    sub     cx, cx
asm    mov     dx, cx        /* zero time */
/* Arrive here with SI = mode, DX:CX = time, and info = directory info. */
sta_construct:
asm    LES     di, bufP
#ifdef ! LDATA
asm    push    DS
asm    pop     ES
#endif
asm    cld
asm    mov     al, info.drive
asm    chw
asm    stosw
asm    xchg    bx, ax        /* device */
asm    sub     ax, ax        /* keep a copy */
asm    stosw
asm    xchg    ax, si        /* inode */
asm    stosw
asm    mov     ax, 1        /* mode */
asm    stosw
asm    xchg    ax, si        /* number of links */
asm    stosw
asm    stosw
asm    xchg    ax, bx        /* bring back the zero */
asm    stosw
asm    stosw
asm    xchg    ax, bx        /* user (owner) id */
asm    stosw
asm    stosw
asm    xchg    ax, bx        /* group id */
asm    stosw
asm    mov     ax, W0 (info.size) /* real device */
asm    stosw
asm    mov     ax, W1 (info.size) /* file.. */
asm    stosw
asm    xchg    ax, cx        /* ..size */
asm    stosw
asm    xchg    ax, dx
asm    stosw
asm    xchg    ax, dx        /* access time */

```

```

asm    stosw asm    xchg    ax, dx
asm    stosw                /* modification time */
asm    xchg    ax, dx
asm    stosw
asm    xchg    ax, dx
asm    stosw                /* status change time */
return 0;
;

```

_status87 取浮点状态

-- stat87.cas

用 法 unsigned int _status87 ();

原 型 在 float.h

说 明 _status87 取浮点状态字，它是 8087/80287 状态字和其它由 8087/80287 例外情况处理程序检测到的条件的组合。

返 回 值 返回值中的位给出浮点状态。对返回位详细定义见 float.h 文件

参 见 _clear87, _control87, _fpreset

源 程 序

```

#pragma inline
#include <float.h>
unsigned int _status87(void)
{
    volatile unsigned    Status;
asm    fstsw    Status
asm    fwait
    return(Status & 0x3F);
}

```

stime 设置时间

-- stime.c

用 法 int stime (long *tp);

原 型 在 time.h

说 明 见 time

源 程 序

```

#include <time.h>
#include <dos.h>
int stime(long *tp)
{
    struct date    d;
    struct time    t;
    unixtodos(*tp, &d, &t);
    setdate(&d);
    settime(&t);
    return (0);
}

```

函数名 _stklen - 堆栈长度。

-- stklen.c

用 法 extern unsigned _stklen;

说 明 定义缺省堆栈长度。

源 程 序

```

#include <asmrules.h>
unsigned    _stklen = 0x1000;    /* Default stack size in bytes */

```

函数名 stl_Digit - 将 ASCII 数字转换为数字。 -- scantol.cas

用 法 static char near stl_Digit (void)

说 明 该过程将'0..9','a..z'/'A..Z'中的字符转换为数字。如果该字符不是数字，则结果为不作修改的字符。

调用时 CL 为基数, 字符在 BL 中。

如果结果为数字, 则清除进位。输出是 AH = AL 的符号扩充不修改其它寄存器。

方法如下所述:

—对于 ASCII <= '9', 将字母'0..9' 映射为值 0..9, 丢弃所有小于'0'的 ASCII 字符;

—对于 ASCII > '9', 将字母'A..Z'或'a..z' 转换为值 10..36, 丢弃所有映射值小于 10 的 ASCII 字符;

—对于正确通过上述映射的值, 如果不小于基数则将其丢弃。

返回值 成功时返回数值; 失败时返回原来字符。

源程序

```
#pragma inline
#include <asmrules.h>
#include <_scanf.h>
#include <stdio.h>
#include <ctype.h>
#pragma warn -rvl
static char near stld_Digit (void)
{
    asm          push    bx          /* remember original character. */
    asm          sub     bl, '0'
    asm          jb      stld_badEnd
    asm          cmp     bl, 9
    asm          jna     stld_digitised /* digitized values are 0..9 */
    asm          cmp     bl, ('Z' - '0')
    asm          ja      stld_maybeLower
    asm          sub     bl, ('A' - '0' - 10)
    asm          jmp     short stld_extended
    stld_maybeLower:
    asm          sub     bl, ('a' - '0' - 10)
    stld_extended:
    asm          cmp     bl, 9          /* extended digits are 10..radix */
    asm          jna     stld_badEnd
    stld_digitised:
    asm          cmp     bl, cl          /* is digit within radix range ? */
    asm          jnb     stld_badEnd
    asm          add     sp, 2          /* forget the original char */
    asm          cld                      /* since we found a true digit */
    asm          mov     bh, 0
    stld_end:
        return;
    stld_badEnd:
    asm          pop     bx          /* recover original character */
    asm          stc                      /* carry set, was not a char. */
    asm          jmp     short stld_end
}
#pragma warn .rvl
```

strcpy 拷贝一个字符串到另一个 -- strcpy.c

用法 char *strcpy (char *destin, char *source);

原型在 string.h

说明 见 str...

源程序

```
#include <string.h>
char *strcpy(char *to, const char *from)
{
    register unsigned len;
    len = strlen(from);
    memcpy(to, from, len+1);
    return (to+len);
}
```

str... 字符串操作系列函数

~ strcat.c
-- strchr.cas
-- strcmp.cas
-- stricmp.cas
-- strlen.cas
-- strlwr.c
-- strncpy.cas
-- strnicmp.cas
-- strrev.c
-- strspn.c
-- strdup.c
-- strerror.c
-- strncat.c
-- strnset.c
-- strpbrk.c
-- strrchr.c
-- strset.c
-- strspn.c
-- strtod.c
-- strtol.c
-- strcpy.cas
-- strncmp.cas
-- strstr.cas
-- strupr.c
-- strtok.c

用 法 char *strcpy(char *destin, char *source);
char *strcat(char *destin, char *source); char *strchr(char *str, char c);
int strcmp(char *str1, char *str2);
char *strcpy(char *str1, char *str2);
int strspn(char *str1, char *str2);
char *strdup char *strdup(char *str);
int stricmp(char *str1, char *str2);
unsigned strlen(char *str);
char *strlwr(char *str);
char *strncat(char *destin, char *source, int maxlen);
int strncmp(char *str1, char *str2, int maxlen);
char *strncpy(char *destin, char *source, int maxlen);
int strnicmp(char *str1, char *str2, unsigned maxlen);
strncmpi(char *str1, char *str2, unsigned maxlen);
char *strnset(char *str, char ch, unsigned n);
char *strpbrk(char *str1, char *str2);
char *strrchr(char *str, char c);
char *strrev(char *str);
char *strset(char *str, char c);

```

int strspn(char *str1, char str2);
char *strstr(char *str1, char *str2);
double strtod (char *str, char **endptr);
long strtol(char *str, char **endptr, int base);
char *strtok(char *str1, char *str2);
char *strupr (char *str);

```

原型在 string.h

说明 以下按字母顺序列出 str...系列函数,接着根据它们的类型或执行任务种类分类介绍这些字符串操作函数,所属类型在括号内注出:

strcat	把一个串加到另一串(合并);
strchr	扫描串中某个给定字符的第一次出现(搜索);
strcmp	把一个串与另一个串进行比较(比较);
strcpy	把一个串拷贝到另一个串(拷贝);
strcspn	扫描串中不包含给定串集合字符的第一个段(比较);
strdup	拷贝串到一个新创建的位置(拷贝);
stricmp	将一个串与另一个串比较,不管大小写(比较);
strncmpi	将一个串与另一个串比较,不管大小写(比较);
strlen	计算串的长度(搜索);
strlwr	转换串中的大写字母为小写(修改);
strncat	把串中的一部分加到另一串(合并);
strncmp	把串中一部分与另一串中一部分比较(比较);
strncpy	拷贝串中的给定字节数到另一串,必要时截断或添加(拷贝);
strncmpi	把串中的一部分与另一个串中的一部分比较,不管大小写(比较);
strnicmp	将串中一部分与另一串中一部分比较,不管大小写(比较);
strnset	将串中指定数目字节设置为给定字符(修改);
strpbrk	扫描给定集合中任一字符在串上第一次出现(搜索);
strchr	扫描给定字符在串中的最后一次出现(搜索);
strrev	颠倒串顺序(修改);
strset	把串中所有字符设置为给定字符(修改);
strspn	扫描给定字符集合的子集在串中第一次出现的段(搜索);
strstr	扫描给定子串在串中的出现(搜索);
strtod	把一串转换为双精度值(转换);
strtok	搜索串中一单词,该单词由第二个串中定义的符合分隔(搜索);
strtol	把一串转换为长整型值(转换);
strupr	把串中所有小写字母转换为大写(修改)。

这 27 个 str...(串操作)函数完成一系列任务,这些任务可以分为六大类:

- 合并
- 修改
- 比较
- 转换
- 拷贝
- 搜索

以下按所完成任务的类型列出每个 str...函数的功能:

1.合并(联结)

strcat 把源串source的拷贝加到目标串destin的末尾, 结果串的长度为`strlen(destin)+strlen(source)`;

strncat 把源串source中最多`maxlen`个字符加到目标串destin后, 再添一空字符。结果串的最大长度为 `strlen(destin)+maxlen`;

2.修改:

strlwr 把串str中的大写字母转换为小写, 无其它变化。

strupr 把串str中的小写字母转换为大写, 无其它变化。

strset 把串str中的所有字符置为字符ch。

strnset 把串str的开始n个字节置为字符ch, 如果`n > strlen(str)`, 那么`strlen (str)`代替n。

strrev 颠倒串中的字符顺序(不包括空字符)。

3.比较:

strcmp 串str1和串str2比较。

stricmp 串str1和串str2比较, 不分大小写。

strcmpi 串str1和串str2比较, 不分大小写(同`stricmp`, 只是它为宏)。

strncmp 同`strcmp`相似, 只是最多比较`maxlen`个字符。

strnicmp 比较串str1和串str2, 最多`maxlen`个字符, 不分大小写。

strcmp 串str1和串str2比较, 最多`maxlen`字符, 不分大小写。

strncmpi 同`strnicmp`, 只是其为宏。

所有这些比较函数都根据 `str1`(或其一部分)与 `str2`(或其一部分)的比较结果返回一个值(`<0,0`, 或`>0`)。

子程序 `strcmpi` 和 `strncmpi` 分别与 `stricmp` 与 `strnicmp` 相同, 只是它们(`strcmpi` 和 `strncmpi`) 是通过 `string.h` 中定义的宏实现的, 这些宏把 `strcmpi` 调用转换为 `stricmp`, 把 `strncmpi` 调用转化为 `strnicmp`. 因此, 为了使用 `strcmpi` 和 `strncmpi`, 必须在文件中包含 `string.h` 头文件以使宏可用。这些宏是为了与其它 C 编译器兼容而提供的。

4.转换:

strtod 把一字符串 str 转换为双精度值。str 为一可以解释为双精度值的字符序列, 格式如下:

`[ws] [sn] [ddd] [.] [ddd] [fmt{sn}ddd]`

其中:

`[ws]` = 可选择的空白字符

`[sn]` = 可选择的符号(+或-)

`[ddd]` = 可选择的数字

`[fmt]` = 可选择的 e 或 E

`[.]` = 可选择的小数点

例如, 以下是 `strtod` 能够转换为双精度值的一些字符串:

`+1231.1981e-1`

`502.85E2`

`-2010.952`

`strtod` 在碰到不能解释为合适双精度值的第一个字符时停止读字符串

如果 `endptr` 不是 `NULL`, 则 `strtod` 把它置为指向停止扫描字符的指针(`*endptr = &stopptr`);

strtol 把一字符串 str 转换为长整型值, str 是一能被解释为长整型值的字符序列, 格式如下:

`[ws] [sn] [0] [x] [ddd]`

其中: [ws]=可选择的空白字符

[sn]=可选择的符号(+或-)

[0]=或选择的零(0)

[x]=可选择的 x 或 X

[ddd]=可选择的数字

strtol 在遇到第一个不认识的字符时停止读字符串

如果 base 在 2 到 36 之间 则长整型数以基数 base 表示。

如果 base 为 0,则 str 的头几个字符决定转换基数:

第一字符 第二字符 字符串解释为

0 1-7 八进制

0 x或X 十六制

1-9 数字

如果 base 为 1, 为无效值

如果 base<0, 为无效值

如果 base>36, 为无效值

任意一个无效值 base 值都使结果变为 0, 且设置下一字符指针为起始串为指针

如果 str 中的值被解释为八进制, 将不认识 0 到 7 外的其它字符。

如果 str 中的值被解释为十进制, 将不认识 0 到 9 外的其它字符。

如果 str 中的值被解释为以其它数作基数, 则只有基数中的数字或字母被认识(如 base=5, 则认识 0 到 4; base=20, 认识 0 到 9,A 到 J)。

5.拷贝:

strcpy 拷贝串source到destin,直到最后一个空字符后停止

strncpy 拷贝串source中maxlen个字符到destin中,有时截断,有时附加空白字符到destin,如果source的长度等于或大于maxlen,目标串destin,不以空字符终结

strcpy 拷贝source中字节到destin,在拷贝完最后一个空字符后停止.strncpy(a,b)与strcpy(a,b)相似,只是返回值不同,strncpy返回a,而strcpy返回a+strlen(b)

strdup 复制串str,通过调用malloc分配空间,大小为(strlen(str)+1)个字节长。

6.搜索

strchr 正向搜索字符串,查找一指定的字符, strchr查找串str中字符ch的第一次出现。空字符终结符被认为是串的一部分,因此:

strchr(strs,0)

将返回指向串strs中终结空字符的指针。

strrchr 反向搜索字符串,查找一指定的字符, strrchr查找串str中字符ch的最后一次出现。空字符终结符也为串的一部分。

strpbrk 扫描串str1,找出串str2中的任一字符的第一次出现。

strspn 返回串str1中包含串str2全部字符的初始段长度。。

strcspn 返回串str1中不包含串str2全部字符的初始段长度。

strstr 扫描串str1中子串str2的第一次出现。

strtok 将串str1看作包含零个或多个文本单词的序列,由分隔符串str2中的一个或多个字符分开。

第一次调用 strtok 时,返回一指向 str1 中第一个单词的第一个字符的指针,并在返回单词后立即写一空字符到 str1 中,后继的使用 NULL 作为第一个参数的调用,将用这种方法扫描串 str1,直到没有剩余单词。

在不同的调用过程中, 分隔字符串 `str2` 可以不同。

当 `str1` 中无单词时, `strtok` 返回 `NULL` 指针。

返回值 以下按函数名的字母顺序列出 `str.` 函数的返回值。

`strcpy` 返回 `destin + strlen(source)`

`strchr` 返回指向串中第一次出现字符 `ch` 的指针。如果 `ch` 在 `str` 中不出现, 返回 `NULL`。

`strcmp`, `stricmp`, `strcmpi`, `strncmp`, `strnicmp` 和 `strncmpi` 返回整型值。

<0 如果 `str1` 小于 `str2`

=0 如果 `str1` 等于 `str2`

>0 如果 `str1` 大于 `str2`

所有这六个函数都完成符号比较。

`strcpy` 返回 `destin`

`strdup` 返回指向包含复制串 `str` 存储位置的指针。在不能分配存储空间时, 返回 `NULL`。

`strlen` 返回 `str` 中的字符数, 不包括空字符终结符

`strncpy` 返回 `destin`

`strpbrk` 返回指向 `str2` 中的任一字符在 `str1` 中第一次出现的指针, 如果 `str2` 中的字符在 `str1` 中都不出现, 返回 `NULL`

`strrchr` 返回指向字符 `ch` 在串中最后一次出现的指针。如果 `ch` 在 `str` 中不出现, 返回 `NULL`

`strrev` 返回指向颠倒顺序的串的指针, 无错误返回。

`strstr` 返回指向串 `str1` 中包含 `str2` 的指针。如果 `str2` 不在 `str1` 中出现, 返回 `NULL`

可移植性 适用于 UNIX 系统, 在 Kernighan 和 Ritchie 书中定义了 `strcat`

参见 `malloc`, `mem...`, `movmem`

源程序

```
#pragma inline
#include <asmrules.h>
#include <string.h>
#pragma warn -use
#pragma warn -rvl
char *strcat(register char *dest, const char *src)
{
    register SI,DI;
    asm cld
    #if defined(__LARGE__) || defined(__COMPACT__)
    asm push ds
    #endif
    #if LDATA
    asm les di, dest /* es:di = dest */
    #else
    asm mov di, dest /* es:di = dest */
    asm push ds
    asm pop es
    #endif
    asm mov dx, di /* save dest offset in dx */
    asm xor al, al
    asm mov cx, -1 /* find end of dest */
    asm repne scasb
    #if LDATA
    asm push es
    #endif
    asm lea si, [di-1] /* es:si points to terminating null in dest */
    asm LES di, src
    asm mov cx, -1 /* figure out strlen(src) */
    asm repne scasb
    asm not cx /* CX = strlen(src) + 1 */
    asm sub di, cx /* point es:di back to start of src */
    #if LDATA
    asm push es
    asm pop ds /* set DS: to seg of src */
    #endif
}
```



```

asm    pop     es                /* restore ES: as seg of dest */
#endif
asm    xchg    si, di            /* DS:SI = src, ES:DI = dest+strlen(dest) */
asm    test    si, 1            /* odd src? */
asm    jz      move_rest
asm    movsb
asm    dec     cx                /* move a byte to make src even */
move_rest:
asm    shr     cx, 1
asm    rep     movsw
asm    jnc     - move_last
asm    movsb
move_last:
asm    mov     ax, dx            /* return addr of string */
#if LDATA
asm    mov     dx, es
#endif
#if defined(__LARGE__) || defined(__COMPACT__)
asm    pop     ds
#endif
return;
}
#pragma warn .use
#pragma warn .rvl
/*****
char *strchr(const char *s, int c)
{
    #if defined(__LARGE__) || defined(__COMPACT__)
    asm    push     ds
    #endif
    asm    LDS     si, s
    asm    mov     bl, c
    asm    test    si, 1            /* SI even? */
    asm    jz      cmp_loop
    asm    lodsb
    asm    cmp     al, bl            /* process first character */
    asm    je      success2
    asm    and     al, al
    asm    jz      failure
    cmp_loop:
    asm    lodsw
    asm    cmp     al, bl            /* get 2 chars at a time */
    asm    je      success1        /* if first char matches */
    asm    and     al, al
    asm    jz      failure        /* give up if end of string */
    asm    cmp     ah, bl
    asm    je      success2        /* if second char matches */
    asm    and     ah, ah
    asm    jnz     cmp_loop        /* continue if more characters */
    failure:
    #if defined(__LARGE__) || defined(__COMPACT__)
    asm    pop     ds
    #endif
    return(0);
    success2:
    asm    inc     si
    success1:
    asm    lea     ax, [si-2]        /* point AX at matching char */
    #if LDATA
    asm    mov     dx, ds
    #endif
    #if defined(__LARGE__) || defined(__COMPACT__)
    asm    pop     ds
    #endif
    return;
}
#pragma warn .rvl
/*****/
int strcmp(const char *str1, const char *str2)

```

```

{
#if defined(__LARGE__) || defined(__COMPACT__)
asm    mov     dx, ds
#endif
#if !(LDATA)
asm    mov     ax, ds
asm    mov     es, ax
#endif
asm    cld
/* Its handy to have AH & BH zero later for the final subtraction. */
asm    xor     ax, ax
asm    mov     bx, ax
/* Determine size of 2nd source string. */
asm    LES     di, str2
asm    mov     si, di
asm    xor     al, al
asm    mov     cx, -1
asm    repne   scasb
asm    not     cx
asm    mov     di, si
asm    LDS     si, str1
/* Scan until either *s2 terminates or a difference is found.      Note that it is
sufficient to check only for right termination, since if the left terminates
before the right then that difference will also terminate the scan. */
asm    repe    cmpsb
/* The result is the signed difference of the final character pair, be they
equal or different. A simple byte subtract and CBW doesn't work here because
it does the wrong thing when the characters are 'ff' and 'ff'.      In that case
255 would be reported as less than 127 as 'ff' sign extends to 'ffh' which
is a negative number.      Remember AH, BH are zero from above. */
asm    mov     al, [si-1]
asm    mov     bl, ES_ [di-1]
asm    sub     ax, bx
#if defined(__LARGE__) || defined(__COMPACT__)
asm    mov     ds, dx
#endif
    return _AX;
}

char *strcpy(char *dest, const char *src)
{
#if !(LDATA)
    _ES = _DS;
#endif
asm    cld
asm    LES     di, src
asm    mov     si, di
asm    xor     al, al
asm    mov     cx, -1
asm    repne   scasb
asm    not     cx
#if !(LDATA)
#if !defined(__HUGE__)
asm    push     DS
#endif
    _DS = _ES;
#endif
asm    LES     di, dest
asm    rep     movsb
#if defined(__LARGE__) || defined(__COMPACT__)
asm    pop     DS
#endif
    return(dest);
}
/*****
size_t strcspn(const char *s1, const char *s2)
{
    register const char *srchs2;
    int len;
    for (len = 0; *s1; s1++, len++)

```

```

        for (srchs2 = s2; *srchs2; srchs2++)
            if (*s1 == *srchs2) goto bye;
bye:
    return (len);
}
/*****
char *strdup(const char *s)
{
    char *p;
    unsigned n;
    n = strlen(s) + 1;
    if ((p = (char *)malloc(n)) != NULL)
        memcpy(p, s, n);
    return (p);
}
*****/
char *strerror(int errnum)
{
    return _maperror(errnum, NULL);
}
/*****
int strcmp(const char *str1, const char *str2)
{
    #if defined(_LARGE_) || defined(_COMPACT_)
    asm mov dx, ds
    #endif
    #if !(LDATA)
    asm mov ax, ds
    asm mov es, ax
    #endif
    ashl cld
    ashl LDS si, str1
    ashl LES di, str2
    /* We setup some constants in registers because there's a slight payoff
       when the strings get longer than 3-4 characters (which should be most
       of the time in a typical program). */
    asm xor ax, ax /* AH and BH stay zero until the end */
    asm mov bx, ax /* when the final sub AX, BX is done */
    asm mov cx, 617aH /* CH = 'a', CL = 'z' */
    cml_nextCh:
    asm lodsb /* AL <- str1[i] */
    asm mov bl, ES_ [di] /* BL <- str2[i] */
    asm or al, al /* null terminator? */
    asm jz cml_end
    asm scasb /* test & advance DI */
    asm je cml_nextCh
    cml_alUpper:
    asm cmp al, ch /* str1[i] < 'a' */
    asm jb cml_blUpper
    asm cmp al, cl /* str1[i] > 'z' */
    asm ja cml_blUpper
    asm sub al, 'a'-'A' /* upper case str1[i] */
    cml_blUpper:
    asm cmp bl, ch /* str2[i] < 'a' */
    asm jb cml_compareAgain
    asm cmp bl, cl /* str2[i] > 'z' */
    asm ja cml_compareAgain
    asm sub bl, 'a'-'A' /* upper case str2[i] */
    cml_compareAgain:
    asm cmp al, bl /* str1[i] == str2[i] */
    asm je cml_nextCh
    cml_end:
    /* We need to do the full word subtract here because ANSI requires unsigned
       comparisons. A simple byte subtract with a CBW would produce the wrong
       result in some cases. Remember that AH, BH are still zero. */
    asm sub ax, bx
    #if defined(_LARGE_) || defined(_COMPACT_)
    asm mov ds, dx
    #endif
}

```

```

        return _AX;
    }
    /*****
size_t  strlen(const char *str)
{
    #if !(LDATA)
        _ES = _DS;
    #endif
    asm    cld
    asm    LES    di, str
    asm    xor    al, al
    asm    mov    cx, -1
    asm    repne scasb
    asm    mov    ax, cx
    asm    not    ax
    asm    dec    ax
    return(_AX);
}
    *****/
    #pragma warn -rvl
    char *strlwr(char *s)
    {
        asm    cld
        #if defined(__LARGE__) || defined(__COMPACT__)
        asm    push    ds
        #endif
        asm    LDS    si, s
        asm    mov    dx, si                /* save addr for return */
        asm    goto    next_char;

    convert_loop:
        asm    sub    al, 'A'                /* see if 'A' .. 'Z' */
        asm    cmp    al, 'Z'-'A'
        asm    ja     next_char
        asm    add    al, 'a'                /* make lowercase */
        asm    mov    [si-1], al
    next_char:
        asm    lodsb
        asm    and    al, al
        asm    jnz    convert_loop
        asm    mov    ax, dx                /* return addr of string */
        #if LDATA
        asm    mov    dx, ds
        #endif
        #if defined(__LARGE__) || defined(__COMPACT__)
        asm    pop    ds
        #endif
    }
    #pragma warn .rvl
    /*****
char *strcat(char *dest, const char *src, size_t maxlen)
{
    register unsigned len;
    unsigned dlen;
    dlen = strlen(dest);
    len = strlen(src);
    if (len > maxlen)
        len = maxlen;
    movmem(src, dest + dlen, len);
    dest[dlen + len] = 0;
    return (dest);
}
    *****/
    int strncmp(const char *str1, const char *str2, size_t maxlen)
    {
        #if defined(__LARGE__) || defined(__COMPACT__)
        asm    mov    dx, ds
        #endif
        #if !(LDATA)
        asm    mov    ax, ds                /* ES = DS */
        asm    mov    es, ax
        #endif
    }

```

```

#endif
asm    cld
/* Determine size of 2nd source string. */
asm    LES    di, str2
asm    mov    si, di
asm    mov    ax, maxlen
asm    mov    cx, ax
asm    jcxz    ncm_end
asm    mov    bx, ax
asm    xor    al, al
asm    repne  scasb
asm    sub    bx, cx
asm    mov    cx, bx
asm    mov    di, si
asm    LDS    si, str1
/* Scan until either *s2 terminates, a difference is found, or "limit" is
reached. Note that it is sufficient to check only for right termination,
since if the left terminates before the right then that difference will
also terminate the scan. */
asm    repe  cmpsb
/* The result is the signed difference of the final character pair, be they
equal or different.
We need to do the full word subtract here because ANSI requires unsigned
comparisons. A simple byte subtract with a CBW would produce the wrong
result in some cases. */
asm    mov    al, [si-1]
asm    mov    bl, ES [di-1]
asm    xor    ah, ah /* zero out high order bytes so the subtraction */
asm    mov    bh, ah /* will work. */
asm    sub    ax, bx
ncm_end:
#if defined(__LARGE__) || defined(__COMPACT__)
asm    mov    ds, dx
#endif
return _AX;
)
/*****/
char *strncpy (char *dest, const char *src, size_t maxlen)
{
#if !(LDATA)
asm    mov    ax, ds
asm    mov    es, ax
#endif
asm    cld
asm    LES    di, src
asm    mov    si, di
asm    xor    al, al
asm    mov    bx, maxlen
asm    mov    cx, bx
asm    repne  scasb
asm    sub    bx, cx
#if (LDATA)
#if !defined ( __HUGE__ )
asm    push    ds
#endif
asm    mov    di, es
asm    mov    ds, di
#endif
asm    LES    di, dest
asm    xchg    cx, bx
asm    rep    movsb
asm    mov    cx, bx
asm    rep    stosb
#if defined ( __LARGE__ ) || defined ( __COMPACT__ )
asm    pop     ds
#endif
return(dest) ;
}
/*****/

```

```

int strcmp(const char *str1, const char *str2, size_t maxlen)
{
    pushD;
#ifdef _LSDATA
    ES = DS;
#endif
    asm cld
    asm LDS    si, str1
    asm LES    di, str2
    asm mov    cx, maxlen
    /* AH and BH will stay zero during this process. This'll be handy when its
       time for a final subtract later. Setting up the 'az' constant in DX can
       have a slight payoff for strings greater than 3-4 chars in length(which
       should be the usual case). */
    asm xor     ax, ax          /* AX <- 0 */
    asm mov     bx, ax          /* BX <- 0 */
    asm mov     dx, 617aH       /* DH <- 'a', DL <- 'z' */
nci_nextCh:
    asm jcxz    nci_end         /* length limited? */
    asm lodsb
    asm mov     bl, ES [di]      /* BL <- str2[i] */
    asm or      al, al          /* end of string? */
    asm jz      nci_end
    asm scasb   /* str1[i] == str2[i]?, advance DI */
    asm loope   nci_nextCh
nci_alUpper:
    asm cmp     al, dh          /* str1[i] < 'a' */
    asm jb      nci_bUpper
    asm cmp     al, dl          /* str1[i] > 'z' */
    asm ja      nci_bUpper
    asm sub     al, 'a' - 'A'    /* upper case str1[i] */
nci_bUpper:
    asm cmp     bl, dh          /* str2[i] < 'a' */
    asm jb      nci_finalDif
    asm cmp     bl, dl          /* str2[i] > 'z' */
    asm ja      nci_finalDif
    asm sub     bl, 'a' - 'A'    /* upper case str2[i] */
nci_finalDif:
    asm cmp     al, bl          /* compare after conversion */
    asm je      nci_nextCh
nci_end:
    /* We need to do the full word subtract here because ANSI requires unsigned
       comparisons. A simple byte subtract with a CBW would produce the wrong
       result in some cases. Remember that AH, BH are still zero. */
    asm sub     ax, bx          /* Get the result. Note: AH & BH are still zero */
    popDS
    return     _AX;
}
/*****
char *strnset(char *s, int ch, size_t n)
{
    unsigned len;
    len = strlen(s);
    if (len < n)
        n = len;
    setmem(s, n, ch);
    return (s);
}
*****/
char *strpbrk(const char *s1, const char *s2)
{
    register const char *srchs2;
    while (*s1)
    {
        for (srchs2 = s2; *srchs2; srchs2++)
            if (*s1 == *srchs2) return((char *)s1);
        s1++;
    }
    return (0);
}
/*****
char *strchr(const char *s, int c)
{
    register const char *ss;

```

```

        for (ss = s + strlen(s); ss >= s; ss--)
            if (*ss == (char)c)
                return ((char *)ss);

        return (0);
    }
}
/*****
#pragma warn -rvl
char *strrev(char *s)
{
    asm    cld
    #if defined(__LARGE__) || defined(__COMPACT__)
    asm    push    ds
    #endif
    asm    LDS     si, s                /* point to string */
    asm    mov     dx, si              /* save addr for return */
    asm    mov     di, si
    asm    push    ds
    asm    pop     es                  /* DS:SI = ES:DI = string */
    asm    xor     al, al
    asm    mov     cx, -1
    asm    repne scasb                 /* find null */
    asm    cmp     cx, -2
    asm    je      reversed            /* abort if string empty */
    asm    dec     di
    asm    dec     di                  /* ES:DI = last non-null char */
    asm    xchg    si, di              /* DS:SI = last, ES:DI = first */
    asm    goto    reverse_entry;
reverse_loop:
    asm    mov     al, [di]
    asm    xchg    al, [si]
    asm    stosb                                     /* swap bytes, bump di */
    asm    dec     si
reverse_entry:
    asm    cmp     di, si              /* pointers crossed? */
    asm    jnb     reverse_loop
reversed:
    asm    mov     ax, dx              /* return addr of string */
    #if LDATA
    asm    mov     dx, ds
    #endif
    #if defined(__LARGE__) || defined(__COMPACT__)
    asm    pop     ds
    #endif
}
#pragma warn rvl
/*****/
char *strset(char *s, int ch)
{
    setmem(s, strlen(s), ch);
    return (s);
}
/*****/
size_t strspn(const char *s1, const char *s2)
{
    register const char *srchs2;
    int len;
    for (len = 0; *s1; s1++, len++)
    {
        for (srchs2 = s2; *srchs2; srchs2++)
            if (*s1 == *srchs2)
                break;
        if (*srchs2 == 0)
            break;
    }
    return (len);
}
/*****/
#if (LDATA)
#define pushES    asm    push    ES
#define popES     asm    pop     ES
#else
#define pushES_

```

```

#define popES_
#endif
char *strstr(const char *str1, const char *str2)
{
    if (!*str2)
        return((char *)str1);
    /* return str1 if str2 empty */

    pushDS_
    #if (LDATA)
        ES = DS;
    #endif
    asm    cld
    asm    LES    di, str1
    pushES_
    asm    mov     bx, di
    asm    xor     ax, ax
    asm    mov     cx, -1
    asm    repnz   scasb
    asm    not     cx
    asm    xchg    cx, dx
    asm    LES     di, str2
    pushES_
    asm    mov     bp, di
    asm    xor     ax, ax
    asm    mov     cx, -1
    asm    repnz   scasb
    asm    inc     cx
    asm    not     cx
    popDS_
    popES_

    strLoop:
    asm    mov     si, bp
    asm    lodsb
    asm    xchg    di, bx
    asm    xchg    cx, dx
    asm    repnz   scasb
    asm    mov     bx, di
    asm    jnz     NotFound
    asm    cmp     cx, dx
    asm    jnb     FirstMatch
    NotFound:
    #if (LDATA)
    asm    xor     bx, bx
    asm    mov     es, bx
    #endif
    asm    mov     bx, 1
    asm    jmp     short End
    FirstMatch:
    asm    xchg    cx, dx
    asm    jecxz   End
    asm    mov     ax, cx
    asm    dec     cx
    asm    repz    cmpsb
    asm    mov     cx, ax
    asm    jnz     strLoop
    End:
    popDS_
    #if (LDATA)
        return (char _es *)(_BX - 1);
    #else
        return (char *)(_BX - 1);
    #endif
}
/*****
double strtod (const char *strP, char **suffixPP)
{
    int    charCt = 0;
    int    status;
    long double result;
    result = _scantod (
        (int *) (void *)Get,
        (void *) (int ch, void *)UnGet,

```



```

        &strP,
        0x7FFF,
        &charCt,
        &status
    );
    if (status <= 0)
        strP -= charCt;
    else if (status == 2)
        errno = ERANGE;
    if (suffixPP != NULL)
        *suffixPP = (char *)strP;
    /*      ldtrunc sets 'errno' to ERANGE if the result
       is to become 0 or HUGE_VAL.      */
    return _ldtrunc(DBL, result, HUGE_VAL);
}
/*-----*/
char *strtok(char *s1, const char *s2)
{
    register const char *sp;
    char *tok;
    if (s1) Ss = (char *)s1;
    /* First skip separators */
    while (*Ss)
    {
        for (sp = s2; *sp; sp++)
            if (*sp == *Ss)
                break;

        if (*sp == 0)
            break;

        Ss++;
    }
    if (*Ss == 0)
        return (0);
    tok = Ss;
    while (*Ss)
    {
        for (sp = s2; *sp; sp++)
            if (*sp == *Ss)
            {
                *Ss++ = 0;
                return (tok);
            }
        Ss++;
    }
    return (tok);
}
/*-----*/
long strtol(const char *strP, char **suffixPP, int radix)
{
    int    charCt = 0;
    int    status;
    long   result;
    errno = 0;
#pragma warn -sus
    result = scantol (
        (int    (*) (void *)) Get,
        (void    (*) (int ch, void *)) UnGet,
        &strP,
        radix,
        0x7FFF,
        &charCt,
        &status
    );
#pragma warn .sus
    if (status <= 0)    strP -= charCt;
    else
        if (status == 2)    errno = ERANGE;
    if (NULL != suffixPP)    *suffixPP = (char *)strP;
    return (result);
}
/*-----*/
char *strupr(char *s)
{

```

```

asm    cld
#if defined(__LARGE__) || defined(__COMPACT__)
asm    push    ds
#endif
asm    LDS     si, s
asm    mov     dx, si          /* save addr for return */
asm    goto    next_char;
convert_loop:
asm    sub     al, 'a'         /* see if 'a' .. 'z' */
asm    cmp     al, 'z' - 'a'
asm    ja      next_char
asm    add     al, 'A'         /* make uppercase */
asm    mov     [si-1], al
next_char:
asm    lodsb
asm    and     al, al
asm    jnz     convert_loop
asm    mov     ax, dx          /* return addr of string */
#if LDATA
asm    mov     dx, ds
#endif
#if defined(__LARGE__) || defined(__COMPACT__)
asm    pop     ds
#endif
}
#pragma warn .rvl

```

_strerror 返回指向错误信息字符串的指针 -- strerror.c

用 法 char *_strerror (const char *string);

原 型 在 stdio.h, string.h

说 明 见 strerror

源 程 序

```

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
char *_strerror(const char *s)
{
    return _maperror(errno, s);
}

```

strerror 返回指向错误信息字符串的指针 -- strerror.c

用 法 char *strerror (int errnum);

原 型 在 stdio.h, string.h

说 明 strerror 返回一个指向与错误号 errnum 相联的错误信息字符串的指针。

_strerror 允许用户生成自己定义的错误信息，它返回一个指向包含错误信息的、以空字符终止的字符串的指针。

如果 str 为 NULL，返回值指向最近产生的系统错误信息，该字符串以空字符终止。

如果 str 不是 NULL，返回值包含 str（或用户定义的错误信息），一个冒号，一个空格，最近产生的系统错误信息和一个换行符。

str 的长度应等于或小于 94 个字符。

strerror 不同于 perror 的地方在于它不打印错误信息。

为更准确地处理错误，应该在（库子程序）出现错误返回时就调用 strerror。

返 回 值 strerror 和 _strerror 都返回一个指向错误信息串的指针。错误信息串被放在一个静态缓冲区中，每次调用 perror 时都被重写。

可移植性 适用于 UNIX 系统

参 见 perror

源程序

```
char *strerror(int errnum)
{
    return _maperror(errnum, NULL);
}
```

函数名 strputn - 复制 n 元素的字符串。 -- sprintf.c

用法 static size_t pascal strputn(char *S, size_t n,
 char **bufPP)

说明 从由 S 指向的数据块中复制 n 个元素到由 bufPP 指向的数据块中，
bufPP 增加 n。

返回值 0。

源程序

```
#include <stdio.h>
#include <mem.h>
#include <string.h>
#include <_printf.h>
static size_t pascal strputn(char *S, size_t n, char **bufPP)
{
    memcpy(*bufPP, S, n);
    *(*bufPP += n) = 0;
    return 0;
}
```

函数名 strtoul - 将字符串转换成无符号的长整数。 -- strtoul.c

用法 unsigned long strtoul(const char *strP, char **suffixPP,
 int radix);

原型文件 stdlib.h

说明 将字符串转换为无符号的长整数。字符串的文法是：

无符号长整数 ::= [空格]*[+]数字；

数字 ::= {'0'['x'['X']][数字]*} | {数字[数字]}

strP 是指向待处理的 ASCII 字符串的指针。suffixPP 是指向待更新的字符串指针的指针。
如果 suffixPP 不为 NULL，则被更新的指针指向已被使用的字符的下一个位置。这样，
调用程序可以简单地分析字符串的后续部分。

radix 可以是 0，或 2、36。如果 radix 为 0，则根据 C 语言中区分八、十或十六进制的规则从 8、10 或 16 中选择基数。

如果 radix > 10 则字母“A..Z”构成合法数字的扩充集合。

返回值

源程序

```
unsigned long strtoul(const char *strP, char **suffixPP, int radix)
{
    int charCt = 0;
    int status = 0;
    long result = 0L;
    while (isspace(*strP))
    {
        strP++;
        charCt++;
    }
    if (*strP != '+')
    {
        errno = 0;
#pragma warn -sus
        result = scantol (
            (int (*) (void *)) Get,
            (void (*) (int ch, void *)) UnGet,
            &strP,
```

```

        radix,
        0x7FFI,
        &charCt,
        &status
    );
#pragma warn .sus
    }
    if (status <= 0)
        strP -= charCt;
    else if (status == 2)
    {
        result = ULONG_MAX;
        errno = ERANGE;
    }
    if (NULL != suffixPP)
        *suffixPP = (char *)strP;
    return (result);
}

```

swab 交换字节 -- swab.c

用 法 void swab (char *from, char *to, int nbytes);

说 明 swab 拷贝字符串 from 的 n 个字节到字符串 to 中，相邻的偶数和奇数字节位置被交换。这在按不同的字节顺序把数据从一台机器送到另一台机器上时是非常有用的。nbytes 应为偶数。

返 回 值 无。

可移植性 适用于 UNIX 系统

源 程 序

```

#pragma inline
#include <asmrules.h>
#define I asm
void swab(char *from, char *to, int nbytes)
{
    I      mov     cx, nbytes           /* BX <- nbytes          */
    I      shr     cx, 1                /* Convert bytes -> words */
    I      jcxz    exit_swab           /* If degenerate case, exit. */
    #if (defined(__COMPACT__) || defined(__LARGE__))
    I      push    ds
    #endif
    I      cld                     /* Make string ops go forward */
    I      LES     di, to              /* ES:DI <- destination */
    I      LDS     si, from            /* DS:SI <- source */
    next_word :
    I      lodsw                     /* Load word from source string */
    I      xchg     ah, al             /* Swap the bytes */
    I      stosw                     /* Store result in destination */
    I      loop     next_word          /* Do the next word */
    #if (defined(__COMPACT__) || defined(__LARGE__))
    I      pop     ds
    #endif
    exit_swab : ;
}

```

system 发出一个 MS-DOS 命令 -- system.c

用 法 int system (char *command);

原 型 在 stdlib.h

说 明 system 通过 MS-DOS 的 COMMAND.COM 文件执行由字符串 command 给定的命令，就象在 DOS 提示符下敲入该命令一样。

COMSPEC 环境变量用于寻找 COMMAND.COM 文件，因此文件不必一定在当前目录中。

返 回 值 system 返回给定的命令完成后，COMMAND.COM 的出口状态。

可移植性 适用于 UNIX 系统。在 Kernig 和 Ritchie 书中作了定义。

参 见 exec..., searchpath, spawn...

源 程 序

```
#include <dos.h>
#include <stdlib.h>
#include <_process.h>
#include <errno.h>
#include <string.h>
int system(const char *cmd)
{
    register char *cmdP;
    int cmdS;
    register char *envP;
    void *envSave;
    char *pathP;
    /* Get COMMAND.COM path */
    if ((pathP = getenv("COMSPEC")) == NULL)
    {
        errno = ENOENT;
        return -1;
    }
    /* Build command line */
    cmdS = 1 + 3 + strlen(cmd) + 1;
    if (cmdS > 128 || (cmdP = malloc(cmdS)) == NULL)
    {
        errno = ENOMEM;
        return (-1);
    }
    if (cmdS == 5)
    {
        cmdP[0] = 0;
        cmdP[1] = '\r';
    }
    else
    {
        *cmdP++ = cmdS - 2;
        *cmdP++ = getswitchchar();
        cmdP = stpcpy(cmdP, "c ");
        cmdP = stpcpy(cmdP, cmd);
        *cmdP++ = '\r';
        cmdP -= cmdS;
    }
    /* Build environment */
    if ((envP = _DOSenv(envron, pathP, &envSave)) == NULL) ,
        errno = ENOMEM;
        free(cmdP);
        return (-1);
    }
    /* Flush all byte streams */
    (* exitbuf)();
    /* Now, call the low level spawn function */
    spawn(pathP, cmdP, envP);
    /* Release all buffers */
    free(envSave);
    free(cmdP);
    return (0);
}
```

tan 三角正切函数

-- tan.cas 用 法 double tan (double x);

原 型 在 math.h

说 明 见 trig

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <math.h>
#include <math.h>
#include <errno.h>
#include <stddef.h>
static unsigned short NANTRIG [4] = {0,0,0x0420, 0x7FF0};
```

```

#pragma warn -rvl
double tan (double x)
{
asm    FLD    DOUBLE (x)
asm    mov    ax, 7FF0h
asm    and    ax, W0 (x [6])
asm    cmp    ax, (53 * 16) + 3FF0h /* extract the exponent field */
asm    jae    tan tooLarge /* biased version of exponent 53 */
asm    if ( _8087 >= 3)
    {
asm    db    OPCODE_FSINCOS
asm    FDIV
    }
    else
    {
asm    FAST_ ( _FTAN )
    }
tan_end:
return;
tan tooLarge: /* total loss of precision */
asm    FSTP   ST(0) /* pop x from stack */
#pragma warn -ret
return _matherr (TLOSS, "tan", &x, NULL, *((double *) NANTRIG));
#pragma warn .ret
}
#pragma warn .rvl

```

tanh 双曲正切函数

-- tanh.cas

用法 double tanh (double x);

原型在 math.h

说明 见 hyperb

源程序

```

#pragma inline
#include <asmrules.h>
#include <math.h>
#include < math.h>
/* Algorithm.
The usual formula is:
    tanh(x) = (exp(x) - exp(-x))/(exp(x) + exp(-x))/
but there is a loss of precision in using this formula directly near 0.
Since tanh(-x) = -tanh(x), compute tanh(|x|) and adjust the sign later
If 0 <= x < 2^-33, return x.
If x >= 32 return 1.
If x >= .17325, use
    y = exp(x)
    tanh(x) = (y - 1/y)/(y + 1/y)
If 2^-33 <= x < .17325, use
    y = exp(2x) - 1
    sinh(x) = y/(2 + y)
where special chip functions are used to get exp(2x)-1 accurately. */
#pragma warn -rvl
double tanh (double x)
{
asm    FLD    DOUBLE (x)
asm    sub    dh, dh
asm    mov    cx, x [6]
asm    shl    cx, 1
asm    rcr    dh, 1 /* DH = sign */
asm    cmp    cx, 8080h
asm    FABS
asm    ja     tanh extreme
asm    cmp    cx, 7F8Ch
asm    jb     tanh small
asm    FAST_ ( _FEXP_ ) /* Exp (x) */
asm    FLD1

```

```

asm    FDIV    st, st(1)
/*    tanh = sinh / cosh = (exp(x) - exp(-x)) / (exp(x) + exp(-x))    */
asm    FLD     st(1)
asm    FSUB    st, st(1)
asm    FXCH
asm    FADD    st, st(2)
asm    FDIVP   st(1), st
asm    FSTP    st(1)
tanh_end:
asm    or      dh, dh
asm    jns     tanh_end2
asm    FCHS
tanh_end2:
    return;
/*    tanh is asymptotic to -1 for negative arguments and +1 for positives.
    It approaches very fast, with exponentially increasing accuracy,
    so it is 1.0 for IEEE accuracy when |x| > 23.    */
tanh_extreme:
asm    FSTP    st(0)          /* pop stack    */
asm    FLD1
    goto tanh_end;
tanh_small:
asm    cmp     cx, 7BC0h
asm    jb      tanh_end
asm    FLD1
asm    FXCH
asm    FSCALE
asm    FLDL2E
asm    FMUL
asm    F2XM1
/* TOS = y = exp(2x) - 1 */
asm    FXCH
asm    FLD1
asm    FADD
asm    FADD    st(0), st(1)
/* stack = 2+y,y */
asm    FDIV
    goto tanh_end;
}
#pragma warn .rvl

```

tell 取文件指针的当前位置 -- tell.c

用 法 long tell (int handle);

原 型 在 io.h

说 明 见 fseek

源 程 序

```

#include <io.h>
long tell(int handle)
{
    return (lseek (handle, 0L, SEEK_CUR));
}

```

textattr 设置文本属性 -- color.c

用 法 void textattr (int attribute);

原 型 在 conio.h

说 明 调用 textattr, 一次就可设置前景和背景颜色 (一般要用 textcolor 和 textbackground 设置它们)。本函数不影响当前屏幕上的任何字符, 它只是影响调用该函数后用直接控制台输出函数 (如 cprintf) 显示的字符。

颜色信息在 attribute 参数中的编码如下:

7 6 5 4 3 2 1 0

B b b b f f f f

在这个 8 位的参数 **attribute** 中:

fff 是 4 位前景颜色 (0 到 15)

bbb 是 3 位背景颜色 (0 到 7)

B 是闪烁允许位

如果闪烁允许打开 (为 on), 字符将会闪烁。这可以通过把属性加上 **BLINK** 来实现。

如果使用在 **CONIO.H** 中定义的符号颜色常量, 通过 **textattr** 来建立文本属性, 请注意时背景颜色选择的如下一些限制:

只能为背景选择前八种颜色之一。

必须将所选的背景颜色左移 4 位, 以使得 3 位背景色移到正确的位置上。

可移植性 只适用 IBM PC 及其与 BIOS 兼容的系统

参 见 **textbackground**, **textcolor**

源 程 序

```
#include <_video.h>
#include <conio.h>
#define INTENSE 0x08
void textattr(int newattr)
{ _video.attribute |= newattr;
}
```

textbackground 选择新的文本背景颜色 -- color.c

用 法 **void textbackground (int color);**

相关函数

用 法 **void textcolor (int color);**

说 明 这些函数为文本和文本背景选择新的颜色。

textcolor 选择前景文本颜色

textbackground 选择背景文本颜色

以后所有由控制台输出函数输出的字符的前景 (或背景) 颜色均由 **color** 确定。这些函数不影响当前屏幕上的任何字符, 只影响调用该函数后用直接控制台输出 (如 **cprintf**) 显示的字符。

对于 **textbackground**, **color** 是从 0 到 7 的一个整型数; 对于 **textcolor**, 则是一个从 0 到 15 的整型数。可以用定义在 **CONIO.H** 中的符号常量来给出颜色, 此时必须使用 **#include <conio.h>**

下表列出了所允许使用的颜色 (符号常量)、它们的值、在前景与背景下有效还是只在前景下有效的情况:

符号常量	数值	前景或背景
BLANK	0	两 者
BLUE	1	两 者
GREEN	2	两 者
CYAN	3	两 者
RED	4	两 者
MAGENTA	5	两 者
BROWN	6	两 者
LIGHTGRAY	7	两 者

DARKGRAY	8	前 景
LIGHTBLUE	9	前 景
LIGHTGREEN	10	前 景
LIGHTCYAN	11	前 景
LIGHTRED	12	前 景
LIGHTMAGENTA	13	前 景
YELLOW	14	前 景
WHITE	15	前 景
BLINK	128	前 景

可以将前景颜色加上 128 来使字符闪烁, 预定义的常量 BLINK 就用于此目的。例如:

```
textcolor (CYAN + BLINK);
```

注意: 有一些监视器不能识别用于产生八种亮度颜色(8-15)的强度信号。在这样的监视器上, 亮度颜色的显示相同于“深度颜色”的显示。另外, 不能显示彩色的系统可能将这些数字看成是一种颜色的灰度、特殊模式或特殊属性(如下划线、黑体、斜体等)。在这样的系统上, 具体的式样要根据系统中的硬件而确定。

返回值 无

可移植性 适用于 IBM PC 及其兼容机, 在 Turbo Pascal 中有相似的程序

参 见 textattr

源 程 序

```
#include <_video.h>
#include <_conio.h>
#define INTENSE 0x08
void textbackground(int newcolor)
{ _video.attribute = (_video.attribute & 0x8F) | ((newcolor << 4) & 0x7f);
}
```

textcolor 在文本模式中选择新的字符颜色 -- color.c

用 法 #include <conio.h>
void textcolor (int color);

原 型 在 conio.h

说 明 见 textbackground

源 程 序

```
#include <_video.h>
#include <_conio.h>
#define INTENSE 0x08
void textcolor(int newcolor)
{ _video.attribute = (_video.attribute & 0x70) | (newcolor & 0x8F);
}
```

textheight 返回以像素为单位的字符串高度

用 法 #include <graphics.h>
int far textheight (char far *textstring);

相关函数

用 法 int far textwidth (char far *textstring);

原 型 在 graphics.h

说 明 textheight 参考当前字体大小和放大因子, 确定以像素为单位的字符串 textstring 的高度。
textwidth 参考字符串长度、当前字体大小和放大因子, 确定以像素为单位的字符串 textstring

的宽度。

这些函数对于调整两行之间的距离大小, 计算视区的高度和宽度、确定一个标题尺寸使其放在图形或方框中的合适位置, 是非常有用的。

例如在 8x8 位图字体和一个为 1 的放大因子 (用 `settextstyle` 设置) 的情况下, 字符串 TurboC 为 8 个像素高, 48 个像素宽。

用 `textheight` 和 `textwidth` 来计算字符串的高度和宽度而不是自己编代码来计算是非常重要的。因为通过使用这些函数后, 即使选择了不同的字体, 也不需要修改源代码。

返回值 `textheight` 返回以像素为单位的字符串高度, `textwidth` 返回以像素为单位的字符串宽度。

可移植性 在 Turbo Pascal 5.0 中有相似的程序。

参 见 `gettextsettings`, `outtext`

textmode 将屏幕设置成文本模式 -- `textmode.c`

用 法 `void textmode (int mode);`

原 型 在 `conio.h`

说 明 `textmode` 选择一个指定的文本模式。

通过使用枚举类型 `text_modes` 的符号常量 (在 `CONIO.H` 中定义) 给出文本模式 (参数 `mode`)。使用这些常量时, 必须使用 `#include <conio.h>`。

`text_modes` 类型的符号常量。数值和所指定的模式见下表:

符号常量	数值	文本模式
LASTMODE	-1	原文本模式
BW40	0	黑白, 40 列
C40	1	彩色, 40 列
BW80	2	黑白, 80 列
C80	3	彩色, 80 列
MONO	7	单色, 80 列

调用 `textmode` 时, 当前窗口被置为全屏, 当前文本属性被置为调用 `normvideo` 后的正常属性。

当 `textmode` 指定 `LASTMODE` 可以使上一次选择的文本方式被重新选择。这种特性只在使用图形模式后想返回到文本模式时才有用。

返回值 无

可移植性 只适用于 IBM PC 及其兼容机, 在 Turbo Pascal 中有相似的程序。

参 见 `gettextinfo`

源 程 序

```
#include <video.h>
#include <conio.h>
void textmode(int newmode)
{
    if (newmode > 3 && newmode != MONO)
        return;
    if (newmode == LASTMODE)
        newmode = _video.currmode;
    crtinit(newmode);
    _video.attribute = _video.normattr;
}
```

textwidth 返回以象素为单位的字符串宽度用 法 `#include <graphics.h>`

`int far textwidth (char far *textstring);`

原 型 在 `graphics.h`

说 明 见 `textheight` 说 明 见 `textheight`

time 取一天的时间

-- `stime.c`

用 法 `long time(long *tloc);`

原 型 在 `time.h`

相关函数

用 法 `int stime(long *tp);`

说 明 `time` 取以秒为单位的, 从 1970 年 1 月 1 月格林威治时间 00:00:00 算起的当前时间, 并把它存在由 `tloc` 所指的位置中。

`stime` 设置系统的时间和日期, `tp` 指向以秒为单位的, 从 1970 年 1 月 1 月格林威治时间 00:00:00 算起的时间值。

返 回 值 `time` 返回如前所述的时间秒值

`stime` 返回 0。

可移植性 适用于 UNIX 系统。

源 程 序

```
#include <time.h>
#include <dos.h>
long time(long *tloc)
{
    struct date    d;
    struct time    t;
    long           x;
    getdate(&d);
    gettime(&t);
    x = dostounix(&d, &t);
    if (tloc)
        *tloc = x;
    return (x);
}
```

tmpfile 以二进制方式打开暂存文件

-- `tmpfile.c`

用 法 `FILE *tmpfile (void);`

原 型 在 `stdio.h`

说 明 `tmpfile` 返回一指向被创建的临时文件流的指针。如果文件不能被创建, 则返回 `NULL`

参 见 `fopen`, `tmpnam`

源 程 序

```
#include <stdio.h>
#include <stdio.h>
FILE *tmpfile(void)
{
    FILE *stream;
    char *s;
    s = tmpnam(NULL); /* Get a unique file name */
    if ((stream = fopen(s, "w+b")) != NULL)
        stream->istemp = _tmpnam;
    return (stream);
}
```

tmpnam 创建唯一的文件名

-- `tmpnam.c`

用 法 `char *tmpnam (char *sptr);`

原 型 在 `stdio.h`

说 明 如果 `sptr` 为 `NULL`, `tmpnam` 返回一指向内部静态对象的指针; 否则返回 `sptr`。

参 见 `tmpfile`

源 程 序

```
#include <stdio.h>
#include <_stdio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
static char    template[L_tmpnam];
unsigned int    tmpnum;
char *tmpnam(char *s)
{
    do
        s = _mkname(s, _tmpnum += (_tmpnum == -1U) ? 2 : 1);
    while (access(s, 0) != -1);
    return (s);
}
```

toascii 转换字符为 ASCII 格式

用 法 `int toascii (int c);`

相关函数

用 法 `int tolower(int c);`
`int toupper(int c);`
`int _tolower(int c);`
`int _toupper(int c);`

原 型 在 `ctype.h`

说 明 `toascii` 通过清除除低 7 位外的所有其它位, 来将整数 `c` 转换为 ASCII 码。其值范围为 0-127, 它是为与其它系统兼容而设计的。

`tolower` 是一用于把整数值 `c` (范围从 EOF 到 255) 转换为小写字母值 (如果原来为大写字母) 而其它不变的函数。

`toupper` 是一用于把整数值 `c` (范围从 EOF 到 255) 转换为大写字母值 (如果原来为小写字母) 而其它不变的函数。

`_tolower` 是一功能同 `tolower` 相似的宏, 但仅用于已知 `c` 是大写字母的情况。

`_toupper` 是一功能同 `toupper` 相似的宏, 但仅用于已知 `c` 是小写字母的情况。

为使用 `_tolower` 和 `_toupper`, 必须包含 `ctype.h` 文件。

返 回 值 在成功时, 每一函数或宏都返回 `c` 的转换值; 出错时不返回值

可移植性 适用 UNIX 系统, `toupper` 和 `tolower` 在 Kernighan 和 Ritchie 书中有定义。

_tolower 把字符转换成小写字母

用 法 `#include <ctype.h>`
`int _tolower(int c);`

原 型 在 `ctype.h`

说 明 见 `toascii`

tolower 把字符转换成小写字母

-- `tolower.c`

用 法 `int tolower (int c);`

原 型 在 `ctype.h`

说 明 见 `toascii`

源 程 序

```
#include <ctype.h>
int tolower(int ch)
{
    if (ch == -1)
        return (-1);
    if (isupper((unsigned char)ch))
        return (_tolower((unsigned char)ch));
    else
        return ((unsigned char)ch);
}
```

_toupper 把字符转换成大写字母

用 法 include <ctype.h>
int _toupper (int c);

原 型 在 ctype.h

说 明 见 toascii

toupper 把字符转换成大写字母

-- toupper.c

用 法 int toupper (int c);

原 型 在 ctype.h

说 明 见 toascii

源 程 序

```
#include <ctype.h>
int toupper(int ch)
{
    if (ch == -1)
        return (-1);
    if (islower((unsigned char)ch))
        return (_toupper((unsigned char)ch));
    else
        return ((unsigned char)ch);
}
```

trig 三角函数

用 法 double acos(double x);
double asin(double x);
double atan(double x);
double atan2(double y, double x);
double cos(double x);
double sin(double x);
double tan(double x);

原 型 在 math.h

说 明 sin, cos 和 tan 返回对应的三角函数值, 角度以弧度表示。

asin, acos 和 atan 各自返回输入值的反正弦、反余弦和反正切值。传给 asin 和 acos 的参数必须在 -1 和 1 之间, 此范围外的值将使 asin 和 acos 返回 0 并置 errno 为:

EDOM 域错误。

atan2 返回 y/x 的反正切值, 即使角度接近 $\pi/2$ 或 $-\pi/2$ (即 x 接近于 0), 也能产生正确的结果。

返 回 值 sin 和 cos 返回一范围在 -1 到 1 之间的值

asin 返回一范围在 $-\pi/2$ 到 $\pi/2$ 之间的值

acos 返回一范围在 0 到 π 之间的值

atan 返回一范围在 $-\pi/2$ 到 $\pi/2$ 之间的值

atan2 返回一范围在 $-\pi/2$ 到 $\pi/2$ 之间的值

tan 返回有效角度的任何值, 对于角度接近于 $\pi/2$ 到 $-\pi/2$ 的情况, tan 返回 0, errno 被置为:

ERANGE 结果超过范围

这些函数的错误处理程序可以通过 matherr 函数修改。

可移植性 适用于 UNIX 系统

参 见 _matherr, matherr, perror

函 数 名 __TrimCtlZ - 删除终结的 Ctrl-Z。 -- zapctrlz.cas

用 法 void pascal __TrimCtlZ (int fildes)

原型文件 _io.h

说 明 如果文件有终结的 Ctrl-Z 则将其删除。

返 回 值 无。

源 程 序

```
void pascal __TrimCtlZ (int fildes)
{
    dosSeekFinalChar (fildes);
asm    push    DX
asm    push    AX          /* remember original position */
    if (dosReadOne (fildes) == ctlZ)
    {
        dosSeekFinalChar (fildes);
asm    mov     bx, fildes
asm    sub     cx, cx /* zero length causes truncation */
asm    sub     dx, dx
asm    mov     ah, 40h
asm    int     21h /* dosWrite (fildes, NULL, 0) */
    }
asm    mov     bx, fildes
asm    pop     dx
asm    pop     cx /* original position */
asm    mov     ax, 4200h + SEEK_SET
asm    int     21h /* DX:AX = lseek (BX, CX:DX, AL) */
    lseek (fildes, 0L, SEEK_SET);
}
```

函 数 名 TrimTrailing - 根据需要压缩尾随的 0。 -- realcvt.cas

用 法 void near TrimTrailing (void);

原型文件 该函数源程序。

说 明 下述规则源于组合 altFormat 和 formCh 参数, 用 backspace 删除不需要的尾随字符。

(altFormat || ~(G or g)) == 需要的尾 0 个数;

altFormat == 所需的小数位数。

返 回 值 在入口, DL 必须包含 formCh, CX 是 strP 的偏移量的开始, ES:[bx]指向字符串尾。

出口处 ES:[bx-1]指向最后一个被删除的字符。

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <_printf.h>
#define I asm
static void near TrimTrailing (void)
{
    I    mov     al, 5Fh          /* "upper case only" mask */
    I    and     al, dl          /* dl is formCh */
    I    cmp     al, 'G'
    I    jne     tt_trimPoint    /* only G/g removes trailing zeroes */
tt_trimLoop:
    I    cmp     BY0 (ES, [bx-1]), '0'
    I    jne     tt_trimPoint
```

```

I      dec     bx
I      cmp     bx, cx
I      ja      tt_trimLoop
I      jmp     short tt_trimmed
tt_trimPoint:
I      cmp     BYTE [ES_ {bx-1}], ' '
I      jne     tt_trimmed
I      dec     bx
tt_trimmed:
    return;
}

```

tzset UNIX 时间兼容函数

-- tzset.cas

用 法 void tzset (void);

原 型 在 time.h

说 明 见 ctime

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <io.h>
#include <io.h>
#include <time.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define YES 1
#define NO 0
#define Normal 0
#define Daylight 1
#define TZstrlen 3 /* Len of tz string(- null terminator) */
#define DefaultTimeZone 5L
#define DefaultDaylight YES
#define DefaultTZname "EST" /* Default normal time zone name */
#define DefaultDSTname "EDT" /* Default daylight savings zone name */
unsigned _monthDay [] =
{ 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365
};
static char DfltZone[ TZstrlen+1 ], DfltLight[ TZstrlen+1 ];
char *const tzname[2] = { &DfltZone[0], &DfltLight[0] };
long timezone = DefaultTimeZone * 60L * 60L; /* Set for EST */
int daylight = DefaultDaylight; /* Apply daylight savings */
void tzset(void)
{
    register int i; /* A loop index */
    char *env; /* Pointer to "TZ" environment string */
#define issign(c) (((c) == '-') || ((c) == '+'))
    if (
        /*****
        1. Check for "TZ" string in the environment.
           env[0] - 1st char in time zone name
           env[1] - 2nd " " " "
           env[2] - 3rd " " " "
           env[3] - 1st char in time zone difference value
           env[4] - 2nd " " " "
        2. Rule out short strings.
        3. Rule out non A-Z time zone characters.
        4. Rule out bad time zone difference numbers.
           a. Not a +/- or 0-9.
           b. Sign with no following digit(s).
        *****/
        /* 1. */ ((env = getenv("TZ")) == 0)
        /* 2. */ (strlen(env) < (TZstrlen+1))
        /* 3. */ ((!isalpha(env[0])) || (!isalpha(env[1])) || (!isalpha(env[2]))) ||
        /* 4a.*/ (!issign(env[ TZstrlen ])) || !isdigit(env[ TZstrlen ]))
        /* 4b.*/ (!isdigit(env[ TZstrlen ]) && (!isdigit(env[ TZstrlen+1 ]))) )

```

```

{
    /*----- Missing or bogus "TZ" string, set default values -----*/
    daylight = DefaultDaylight;
    timezone = DefaultTimeZone * 60L * 60L;
    strcpy(tzname[Normal], DefaultTZname);
    strcpy(tzname[Daylight], DefaultDSTname);
}
else /*----- Parse the "TZ" string and set values from string -----*/
{
    memset(tzname[Daylight], 0, TZstrlen+1); /* Dlt daylight to NULL */
    strncpy(tzname[Normal], env, TZstrlen); /* Set zime zone string */
    tzname[Normal][TZstrlen] = '\0'; /* Force NULL termination*/
    timezone = atol(&env[TZstrlen]) * 3600L; /* Base timezone on "TZ" */

    /*----- Scan for optional daylight savings field -----*/
    /* Scan for string start */
    for (daylight=NO,i=TZstrlen; env[i]; i++)
    {
        if (isalpha(env[i])) /* Found the string start */
        {
            if ((strlen(&env[i])<TZstrlen) ||
                (!isalpha(env[i+1])) ||
                (!isalpha(env[i+2])) )
                break;
            /* Copy and null-terminate dlt sav string */
            strncpy(tzname[Daylight], &env[i], TZstrlen);
            tzname[Daylight][TZstrlen] = '\0';
            daylight = YES;
            break;
        }
    }
}
}

```

ultoa 转换一个无符号长整型数为字符串 -- itoa.cas

用 法 char *ultoa (unsigned long value, char *string, int radix);

原 型 在 stdlib.h

说 明 见 itoa

源 程 序

```

#pragma inline
#include <asmrules.h>
#include <stdlib.h>
#include <_printf.h>
#include <rules.h>
char *ultoa (unsigned long value, char *strP, int radix)
{
    return __longtoa (value, strP, radix, false, 'a');
}

```

函数名 umask - 设置文件读/写权限屏蔽。 -- umask.c

用 法 #include <io.h>

unsigned umask(unsigned modeMask)

原型文件 io.h

说 明 修改用于 open 和 creat 调用的读/写权限屏蔽。

返 回 值 被替代的 umask 值。无错误返回。

注 意 为了方便, umask 被颠倒存放。

源 程 序

```

#include <io.h>
extern unsigned notUmask;
unsigned umask(unsigned modeMask)

```



```

{
    register oldMask;
    oldMask = ~ notUmask;
    notUmask = ~modelMask;
    return (oldMask);
}

```

函数名 `UnGet` - 将字符指针往回移一个位置。 -- `strtok.c`

用法 `static void UnGet(char c, char **strPP);`

说明 字符指针减一。

源程序

```

static void UnGet(char c, char **strPP)
{
    --(*strPP); /* ignore c, we don't allow the string to change */
    #pragma warn -par
}
#pragma warn .par

```

`ungetc` 把一个字符送回输入流中 -- `ungetc.c`

用法 `#include <stdio.h>`

`int ungetc (char c, FILE *stream);`

原型在 `stdio.h`

说明 见 `getc`

源程序

```

#include <stdio.h>
#undef ungetc /* remove macro version */
int ungetc(int c, FILE *fp)
{
    if (EOF != c)
        if (++(fp->level) > 1)
            return (unsigned char) (--(fp->curp) = c);
        else if (1 == fp->level) /* buf was empty, use hold */
        {
            fp->curp = &fp->hold;
            return (unsigned char) (fp->hold = c);
        }
        else
            fp->level--; /* file was not in input mode */
    return (EOF);
}

```

`ungetch` 把一个字符送回键盘缓冲区中 -- `getch.cas`

用法 `int ungetch (int c);`

原型在 `conio.h`

说明 见 `getc`

源程序

```

int ungetch(int c)
{
    if (_cFlag)
        return(EOF);
    _cFlag = 1;
    _AX = c;
    _cChar = AL;
    return _AX;
}

```

`unixtodos` 把日期和时间转换成 DOS 格式 -- `timecvt.c`

用法 `#include <dos.h>`

`void unixtodos (long utime, struct date *dateptr,`

struct time *timeptr);

原型在 dos.h

说 明 见 dostounix

源 程 序

```
#include <io.h>
#include <dos.h>
#include <time.h>
static char Days[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
void unixtodos(long time, struct date *d, struct time *t)
{
    tzset(); /* get timezone info */
    time -= 24L * 60L * 60L * 3652L + timezone;
    t->ti_hund = 0;
    t->ti_sec = time % 60;
    time /= 60; /* Time in minutes */
    t->ti_min = time % 60;
    time /= 60; /* Time in hours */
    d->da_year = 1980 + (int)((time / (1461L * 24L)) < 2);
    time %= 1461L * 24L;
    if (time > 366 * 24)
    {
        time -= 366 * 24;
        d->da_year++;
    }
    #pragma warn -sig
    d->da_year += time / (365 * 24);
    time %= 365 * 24;
    if (daylight && isDST( time % 24, time / 24, 0, d->da_year-1970 ))
        time++;
    #pragma warn .sig
    t->ti_hour = time % 24;
    time /= 24; /* Time in days */
    time++;
    if ((d->da_year & 3) == 0)
    {
        if (time > 60)
            time--;
        else
            if (time == 60)
            {
                d->da_mon = 2;
                d->da_day = 29;
                return;
            }
    }
    for (d->da_mon = 0; Days[d->da_mon] < time; d->da_mon++)
        time -= Days[d->da_mon];
    d->da_mon++;
    d->da_day = time;
}
```

unlink 删掉一个文件

- unlink.cas

用 法 int unlink (char *filename);

相关函数

用 法 int remove (char *filename);

原 型 在 dos.h

说 明 unlink 删掉由 filename 所指定的文件。filename 中可以包含任意 MS-DOS 驱动器、路径和文件名，不允许有匹配符。

只读文件不能通过这个调用删除。为了删除只读文件，必须首先使用 chmod 或 _chmod 来改变属性。

remove 是一简单地把调用传给 unlink 的宏。

返 回 值 在调用成功时，返回 0；出错时，返回 -1，并置 errno 为下列值之一。

ENOENT 路径或文件名没有找到

EACCES 无此权限

可移植性 适用于 UNIX 系统

参 见 chmod

源 程 序

```
#pragma inline
#include <asmrules.h>
#include <dos.h>
#include <io.h>
int unlink(const char *filename)
{
    pushDS
    asm    mov     ah, 041h
    asm    LDS     dx, filename
    asm    int     021h
    popDS
    asm    jc      unlinkFailed
    return(0);
unlinkFailed:
    return _IOerror(_AX);
}
```

unlock 解除文件共享锁

-- lock.cas

用 法 int unlock(int handle, long offset, long length);

原 型 在 dos.h

说 明 见 lock

源 程 序

```
#pragma inline
#include <io.h>
#include <io.h>
int unlock(int handle, long offset, long length)
{
    asm    mov     ax, 05C01h
    asm    mov     bx, handle
    asm    mov     cx, offset+2
    asm    mov     dx, offset
    asm    mov     si, length+2
    asm    mov     di, length
    asm    int     021h
    asm    jc      unlockFailed
    return(0);
unlockFailed:
    return _IOerror(_AX);
}
```

va... 实现可变的参数表

用 法 #include <stdarg.h>

```
void va_start(va_list param, lastfix);
type va_arg(va_list param, type);
void va_end(va_list param);
```

原 型 在 stdarg.h

说 明 有些 C 函数，如 `vfprintf` 和 `vprintf`，用可变的参数表而不是固定数目（已知）的参数表。
va...宏提供了一种存取这些参数表的方便方法，当被调用函数不知道参数的个数和类型时，它们可以用于测试参数表。

头文件 `stdarg.h` 中说明了一个类型 (`va_list`) 和三个宏 (`va_start`, `va_arg` 和 `va_end`)。

1. `va_list`

本数组用于存放 `va_arg` 和 `va_end` 所需要的信息。当一被调用函数使用一个可变的参数

表时, 它说明一个类型为 `va_list` 的变量 `param`。

2. `va_start`

本子程序 (用宏实现) 使 `param` 指向被传送给函数的可变参数表的第一个参数。在调用 `va_arg` 和 `va_end` 之前, 必须先调用 `va_start`。

`va_start` 接受两个参数: `param` 和 `lastfix`, (`param` 已在前段 `va_list` 中解释, `lastfix` 为传递给被调用函数的最后一个固定参数的名)

3. `va_arg`

本子程序 (也用宏实现) 扩展表达式使其与下一个被传递参数具有相同的类型和值。

`va_arg` 中的变量 `param` 应与 `va_start` 初始化时的 `param` 相同。

在第一次使用 `va_arg` 时它返回表中的第一个参数, 后续的每次调用都返回表中的下一个参数。这是通过先访问 `param`, 然后把它增加以指向下一项而实现的。`va_arg` 使用 `type` 来完成访问和定位下一项。每调用一次 `va_arg`, 它都修改 `param` 以指向表中的下一参数。

4. `va_end`

本宏用于被调用函数完成一正常返回。它可以修改 `param` 使其在重新调用 `va_start` 以前不能被使用。`va_end` 必须在 `va_arg` 读完所有的参数后才被调用, 否则会产生意想不到的情况。

返回值 `va_start` 和 `va_end` 不返回值; `va_arg` 返回表中的当前参数 (`param` 所指项)

可移植性 适用于 UNIX 系统。

参 见 `...scanf`, `printf`

`va_arg` 存取可变参数表

用 法 `#include <stdarg.h>`

`type va_arg (va_list param, type);`

原型在 `stdarg.h`

说 明 见 `va_...`

`va_end` 结束可变参数存取

用 法 `#include <stdarg.h>`

`void va_end(va_list param);` 原型在 `stdarg.h`

说 明 见 `va_...`

函数名 `__validatexy` - 检查屏幕坐标的合法性。-- `screen.c`

用 法 `int pascal __validatexy(int x1, int y1, int x2, int y2);`

返回值 如果坐标非法则返回 0。

源程序

```
int pascal __validatexy(int x1, int y1, int x2, int y2)
{
    _CX = _video.screenwidth;
    _DX = _video.screenheight;
    return !(x1 > _CX || x2 > _CX || x1 > x2 ||
            y1 > _DX || y2 > _DX || y1 > y2);
}
```

`wva_start` 开始可变参数存取

用 法 `#include <stdarg.h>`

`void wva_start (va_list param, lastfix);`

原型在 `stdarg.h`

说明 见 `va_...`

`vprintf` 送格式化输出到一流中

-- `vprintf.c`

用法 `#include <stdio.h>`

`#include <stdarg.h>`

`int vprintf (FILE *stream, char *format, va_list param);`

原型在 `stdio.h`, `stdarg.h`

说明 见 `printf`

源程序

`#include <stdio.h>`

`#include <_printf.h>`

`#include <stdio.h>`

`int cdecl vprintf (FILE *F, const char *fmt, va_list ap)`

`{`
`return __vprinter ((putc_t *) _putc, F, fmt, ap);`
`}`

`vscanf` 从流中执行格式化输入

-- `vscanf.c`

用法 `#include <stdio.h>`

`int vscanf (FILE *stream, char *format, va_list param);`

原型在 `stdio.h`

说明 见 `...scanf`

源程序

`#include <stdarg.h>`

`#include <stdio.h>`

`#include <scanf.h>`

`#undef ungetc /* remove the macro version */`

`int cdecl vscanf (FILE *fp, const char *fmt, va_list ap)`

`{`
`#pragma warn -sus`
`return scanner ((int (*) (void *)) fgets,
 (void (*) (int ch, void *)) ungetc,
 fp,
 fmt,
 ap
);
 #pragma warn .sus
}`

`_VideoInt` 产生视频中断。

-- `crtime0.cas`

用法 `void _VideoInt(void);`

原型文件 `_video.h`

说明 通过 DOS 中断 10 产生视频中断。

源程序

`#pragma saveregs`

`void _VideoInt(void)`

`{`
`static unsigned oldbp;`
`DI = DI;`
`oldbp = BP;`
`genInterrupt(0x10);`
`BP = oldbp;`
`}`

函数名 `__vprinter` - 格式化输出。 -- `vprinter.cas`

用法 `int pascal __vprinter (putnF *putter,
void *outP,
const char *formP,
va_list argP)`

原型文件 `_printf.h`

说明 参数序列*argP由相应于格式串*formP的正文文字组成。

过程*putter用于产生输出,它要求有由__vprinter构造的字符串S,并将其复制到目标destP。目标可以是一个字符串、一个文件或调用程序所需要的构造类型。由于缓冲区具有限定的大小,因此对*putter可以有几个调用。

*putter用于保留SI,DI。

使用参数outP的唯一目的是将其传递给putter。

位于*argP的目标是不知结构的记录,在格式串的帮助下对其结构进行解释。该结构的每个域可以是整数、long、double或字符串(char*)。字符可以出现,但都占据一个整数单元。浮点数(float)、字符数组和结构可能不出现。

返回值 该函数的结果是传递给*outp的字符个数。

这里没有错误指示。遇到非法转换时,__vprinter将格式串作为文字进行复制(因为这里假设参数序列的对应已经失去),开头的%引入错误的格式。

如果目标outP大小有限,例如一个字符串或整个磁盘,__vprinter并不知道。目标的溢出引起结果未定义。在有些情况下,*putter可以安全地处理溢出,但这并非__vprinter所关心的。

格式串的语法是:

```
format ::= ((literal) {'%' conversion }) * ;  
conversion ::= '%' | [flag]* [width] ['.' precision]  
               ['t'] type ;  
flag ::= '.' | '+' | '-' | '#' | '0' ;  
width ::= '*' | number ;  
precision ::= '.' ('*' | number) ;  
type ::= 'd' | 'i' | 'o' | 'u' | 'x' | 'X' | 'r' | 'e' | 'E' |  
         'g' | 'G' | 'c' | 's' | 'p' | 'N' | 'f'
```

源程序

```
#pragma warn -use  
int pascal __vprinter (putnF *putter,  
void *outP,  
const char *formP,  
va_list argP)  
{  
#define Ssize 80  
typedef  
enum  
{  
flagStage, fillzStage, wideStage, dotStage, precStage,  
ellStage, typeStage,  
} syntaxStages;  
typedef  
enum  
{  
altFormatBit = 1, /* the '#' flag */  
leftJustBit = 2, /* the '.' flag */  
notZeroBit = 4, /* 0 (octal) or 0x (hex) prefix */  
fillZerosBit = 8, /* zero fill width */  
isLongBit = 16, /* long-type argument */  
farPtrBit = 32, /* far pointers */  
}
```

```

        alt0xBit    = 64,      /* '#' confirmed for %x format */
        floatBit    = 128,    /* float arg 4 bytes not 8! */
        LongDoubleBit = 256   /* signal a long double argument */
    } flagBits;

flagBits flagSet;
ord16 aP;
char fc;
char isSigned;
int16 width;
int16 precision;
char plusSign;
int leadZ;
ord16 abandonP;
char tempStr ( _CVTMAX_ );
card16 totalSent = 0;
card8 Scount = Ssize;
char S [Ssize];
int fit_arg_len;
register SI, DI;
/* the remaining variables are held entirely in registers */
char hexCase;
long tempL;
syntaxStages stage; -- CH
char c;
char *cP;
int *iP;

#endif
#if 0
/* Warning: the following C code is comment only ! It has not been tested. */
    aP = 0;
#define PutToS(c) ( \
    S[aP++] = c; \
    if (--Scount == 0) { \
        S[aP] = 0; \
        putter (S, Ssize, outP); \
        aP = 0; \
        totalSent += (Scount = Ssize); \
    } \
)
vpNEXT:
    if ('\0' == (fc = *(formP++))) /* the normal end */
    {
        if (Ssize - Scount)
        {
            totalSent += (Ssize - Scount);
            putter (S, Ssize - Scount, outP);
        }
        return totalSent;
    }
    if (('%' == fc) && ('%' != (fc = *(formP++))))
        goto vpCONV;
    PutToS (fc);
    goto vpNEXT;
vpCONV:
    abandonP = (unsigned) formP;
    width = -1;
    precision = -1;
    plusSign = '\0';
    leadZ = 0;
    #if LDATA
        flagSet = farPtrBit;
    #else
        flagSet = 0;
    #endif
    stage = flagStage;
    cP = 1+tempStr; /* tempStr [0] may be used for inserting '+' */
    goto vpDoSwitch;
vpNextSwitch:
    fc = *(formP++);

```

```

vpDoSwitch;
if ((fc < ' ') || (fc & 0x80))
    goto vpAbandon;
switch (printCtype [fc])
{
    case (_af): /* alternateForm */
        if (stage > flagStage)
            goto vpAbandon;
        flagBits |= altFormatBit;
        goto vpNextSwitch;
    case (_lj): /* leftJust */
        if (stage > flagStage)
            goto vpAbandon;
        flagBits |= leftJustBit;
        goto vpNextSwitch;
    case (_st): /* sign fill */
        if (stage > flagStage)
            goto vpAbandon;
        if (plusSign != '+')
            plusSign = fc;
        goto vpNextSwitch;
    case (_ne): /* near pointer */
        flagBits &= ~farPtrBit;
        stage = ellStage;
        goto vpNextSwitch;
    case (_fa): /* far pointer */
        flagBits |= farPtrBit;
        stage = ellStage;
        goto vpNextSwitch;
    case (_ar): /* format by arg */
        temp = (((int *) argP)++);
        if (stage < wideStage)
        {
            width = temp;
            stage = wideStage + 1;
        }
        else if (stage == precStage)
        {
            precision = temp;
            stage++;
        }
        else
            goto vpAbandon;
        goto vpNextSwitch;
    case (_fz): /* fillZeros */
        if (stage > flagStage)
            goto vpr NUMERAL;
        if (flagBits & leftJustBit)
            goto vpNextSwitch;
        flagSet |= fillZerosBit;
        stage = fillzStage;
        goto vpNextSwitch;
    case (_pr): /* precision 'r' */
        if (stage >= precStage)
            goto vpAbandon;
        stage = precStage;
        goto vpNextSwitch;
    case (_nu): /* numeral */
        if (stage <= wideStage)
        {
            width = (width < 0) ? fc - '0' :
                width * 10 + (fc - '0');
            stage = wideStage;
        }
        else if (stage != precStage)
        {
            precision = precision * 10 +
                (fc - '0');

```



```

        stage = precStage;
    }
    else
        goto vpAbandon;
    goto vpNextSwitch;
case (_lo): /* long */
    flagSet |= isLongBit;
    stage = ellStage;
    goto vpNextSwitch;
case (_ld): /* long double */
    flagSet |= LongDoubleBit;
    stage = ellStage;
    goto vpNextSwitch;
case (_sh): /* short */
    flagSet &= ~isLongBit;
    stage = ellStage;
    goto vpNextSwitch;
case (_de) :
    radix = 10;
    goto vpINT;
case (_oe) :
    radix = 8;
    goto vpUINT;
case (_un) :
    radix = 10;
    goto vpUINT;
case (_he) :
    hexCase = fc - ('x' - 'a');
    radix = 16;
    goto vpUINT;
case (_fl) :
    goto vpFLOAT;
case (_ch) :
    cP = (char *) (((int *) argP)++);
    cP[1] = 0;
    goto vpCOPY;
case (_st) :
    cP = *(((char **) argP)++);
    goto vpCOPY;
case (_ns) :
    /* number sent */
    iP = *(((int *) argP)++);
    *iP = totalSent + Ssize - Scount;
    goto vpNEXT;
case (_pt) :
    goto vpPointer
case (_zx) :
case (_pc) :
case (_dc) :
    goto vpAbandon;
}

vpUINT:
    isSigned = false;
    templ = (flagSet & isLongBit) ? *(((long *) argP)++) :
        *(((unsigned *) argP)++);
    goto vpPUTINT;
vpINT:
    isSigned = true;
    templ = (flagSet & isLongBit) ? *(((long *) argP)++) :
        *(((int *) argP)++);
vpPUTINT:
    notZero = templ != 0L;
    cP = 1 + tempStr; /* tempStr[0] reserved for sign */
    if ((precision == 0) && (!notZero)) /* ANSI says result is no chars */
    {
        char fillChar; int i;
        if ((width != 0) && (width != -1))
        {
            fillChar = (flagSet & fillZerosBit) ? '0' : ' ';
            for (i=0; i < width; i++)

```

```

        tempStr[i] = fillChar;
    }
    goto vpr_NEXT;
}
longtoa (tempL, cP, radix, isSigned, hexCase);
if (precision > 0)
{
    /* Can't drop digits(ANSI), widen it if necessary */
    if (precision > (len = strlen (cP) - (*cP == '.')))
        leadZ = precision - len;
    else
        precision = len;
}
goto vpNUMERIC;
vpPointer:
isSigned = false;
cP = tempStr;
tempL = *(((unsigned *) argP)++);
if (flagSet & farPtrBit)
{
    Hex4 (cP, *(((unsigned *) argP)++));
    cP += 4;
    *(cP++) = '.';
}
Hex4 (cP, tempL);
tempL = (flagSet & farPtrBit) ? *(((long *) argP)++) :
    *(((unsigned *) argP)++);
notZero = false;          /* suppress check for 0/0x/0X prefixing */
*(cP += 4) = '\0';
len = cP - tempStr;
cP = tempStr;
precision = MAX (len, precision);
goto vpCOPY;
vpFLOAT:
cP = 1 + tempStr;          /* tempStr [0] reserved for sign */
__realcvt (((double *) argP)++, (precision > 0) ? precision : 6,
    cP, fc, altFormat);
notZero = false;          /* suppress check for 0/0x/0X prefixing */
goto vpCOPY;
vpNUMERIC:
if (plusSign && (*cP != '.'))
    * (--cP) = plusSign;
vpCOPY:
len = strlen (cP);
vpr_CopyLen:
if (altFormat & notZero)
{
    if ((fc == 'o') && (leadZ <= 0))
        leadZ = 1;
    if ((fc == 'x') || (fc == 'X'))
    {
        flagSet |= alt0xbit;
        width -= 2;
        if ((leadZ - 2) < 0)
            leadZ = 0;
    }
}
if (! leftJust)
    while (width > (len + leadZ))
    {
        width--;
        PutToS (' ');
    }
if ((flagSet & notZeroBit) && (flagSet & alt0xBit))
{
    PutToS ('0');
    PutToS (fc);
}
if (leadZ > 0)

```

```

    {
        len -= leadZ;
        width -= leadZ;
        if (((c = *cP) == ',') || (c == ' ') || (c == '+'))
            if (len > 0)
            {
                width--;
                len--;
                PutToS (*(cP++));
            }
        while (leadZ-- > 0)
            PutToS ('0');
    }
vpr_actualCopy:
    width -= len;
vpr_copyLoop:
    while (len -- > 0) /* this is the high-point of __vprinter ! */
        PutToS (*(cP++));
    if (leftJust)
        while (width-- > 0)
            PutToS (' ');
    goto vpNEXT;
#endif
I jmp short func_start /* Skip over inline nested PROCs */
/*****
    local, nested functions are placed here
    ES *must* be defined in all models.
*****/
#define RETN db 0xC3 /* "RETN" is used to force a near ret */
/* Get length of string pointed to by ES:DI, return result in CX. */
I vpr_strlen LABEL NEAR /* scan string ES: [DI] up to \0 */
I push di
I mov cx, -1 /* count the string length. */
I xor al, al
I repne scasb
I not cx /* (not CX) == (-1 -CX) */
I dec cx /* scasb overshoots */
I pop di
I RETN
/* call *putter to flush S */
/* Put character to next position in S, check for S full */
I vpr_PutToS LABEL NEAR
I mov BY0 (SS_ [di]), al
I inc di
I dec BY0 (Scount)
I jle exit_PutToS
I vpr_CallPutter LABEL NEAR
I push bx
I push cx
I push dx
I push ES
I lea ax, S
I sub di, ax /* count chars in S */
I putter (S, DI, outP);
I Scount = Ssize;
I totalSent += _DI;
I lea di, S
I pop ES
I pop dx
I pop cx
I pop bx
exit_PutToS :
I RETN /* 'shared' RET for both PROCs */
/***** end of embedded functions *****/
func_start:
I push ES
I cld
I lea di, S
I mov sp, di

```

```

/*
This paragraph is arranged to give in-line flow to the most frequent
case, literal transcription from *formP to *outP.
*/
vpr_NEXTap:
I      mov     di, ap
vpr_NEXT:
I      LES     si, formP          /* loop to here when DI still valid */
I      for (;;)                  /* resume here from this literal/space */
{
vpr_nextCh :
I      lods    BY0 (ES_ [si])    /* if (pattern[x] == '\0') */
I      or      al, al            /* goto exit */
I      jz      vpr_respondJump
I      cmp     ai, '%'           /* '%' char begins a conversion */
I      je      vpr_CONV         /* but "%%" is just a literal '%'. */
vpr_literal:
I      mov     SS_ [di], al
I      inc     di

I      dec     BY0 (Scout)       /* If (--Scout) */
I      jg      vpr_nextCh       /* continue; */

#ifdef FASTWAY
I      push    ES                /* Fast way (~ 1-2% speed gain) */
I      lea     ax, S
I      sub     di, ax            /* count chars in S */
I      putter (S, DI, outP);
I      Scout = Ssize;
I      totalSent += DI;
I      lea     di, S
I      pop     ES
#else
I      call    vpr_Compacter     /* Compact way (smaller) */
#endif
}

vpr_respondJump:
I      jmp     vpr_respond
/* If arrived here then a conversion specification has been encountered. */
vpr_CONV:
I      mov     W0 (abandonP), si /* abandon will print from here */
I      lods    BY0 (ES_ [si])    /* AL <- char at ES:SI */
I      cmp     al, '%'           /* %% really means % */
I      je      vpr_literal
I      mov     W0 (ap), di       /* keep result pointer safe */
I      xor     cx, cx            /* CH is flagStage */
I      mov     W0 (leadZ), cx    /* leadZ <- 0 */
#ifdef LDATA
I      mov     W0 (flagSet), farPtrBit /* flagSet <- pointers are FAR */
#else
I      mov     W0 (flagSet), cx    /* flagSet <- 0 */
#endif
I      mov     BY0 (plusSign), cl /* plusSign <- 0 */
I      mov     W0 (width), -1     /* width <- default */
I      mov     W0 (precision), -1 /* precision <- default */
I      jmp     short vpr_doSwitch
/*=====*/
/* loop to here when scanning flags */
/*=====*/

vpr_nextSwitch:
I      lods    BY0 (ES_ [si])    /* AL <- char at ES:SI */
/*=====*/
/* Main character classification switch
/*=====*/

vpr_doSwitch:
I      xor     ah, ah            /* Remove any high order trash */
I      mov     dx, ax            /* Duplicate original char in DL */
I      mov     bx, ax            /* Duplicate original char in BL */
I      sub     bl, ' '           /* Scale char in BL to 0-95 */
I      cmp     bl, 128           /* Weed out don't cares */

```

```

I    jae    vpr_jumpAbandon
I    mov    bl, BY0 (printCtype [bx]) /* BL <- char type of the char */
I    switch (BX) /* ==> clobbers AX, BX <== */
{
vpr_jumpAbandon: /* Extend local jump range */
I    jmp    vpr_abandon
case (_af): /* when '#' was seen */
I    cmp    ch, flagStage
I    ja     vpr_jumpAbandon
I    or     W0 (flagSet), altFormatBit
I    jmp    vpr_nextSwitch
case (_lj): /* when 'l' was seen */
I    cmp    ch, flagStage
I    ja     vpr_jumpAbandon
I    or     W0 (flagSet), leftJustBit
I    jmp    vpr_nextSwitch
case (_st): /* when 's' or 't' was seen */
I    cmp    ch, flagStage
I    ja     vpr_jumpAbandon
I    cmp    BY0 (plusSign), 2Bh /* '+'
I    je     gtyp_nxt_swit /* ' ' ignored if '+' already */
I    mov    plusSign, dl
gtyp_nxt_swit:
I    jmp    vpr_nextSwitch
case (_ne): /* near pointer */
I    and    W0 (flagSet), NOT farPtrBit
I    mov    ch, ellStage
I    jmp    vpr_nextSwitch
case (_fa): /* far pointer */
I    or     W0 (flagSet), farPtrBit
I    mov    ch, ellStage
I    jmp    vpr_nextSwitch
case (_fr): /* leading width '0' is a flag */
I    cmp    ch, flagStage
I    ja     case_nu /* else it is just a digit */
I    test   W0 (flagSet), leftJustBit
I    jnz    short vpr_nextSwitchJmp
I    or     W0 (flagSet), fillZerosBit
I    mov    ch, fillZStage /* but it must be part of width */
I    jmp    vpr_nextSwitch
vpr_abandonJmp: /* Extend local jump range */
I    jmp    vpr_abandon
case (_ar): /* when '*' was seen it causes */
#ifdef LDATA
I    push    ES
#endif
I    LES     di, argP
I    mov     ax, ES [di] /* the next argument to be
I    add     W0 (argP), 2 /* taken, depending on
#ifdef LDATA
I    pop     ES
#endif
I    cmp     ch, wideStage /* the stage, as the
I    jnb     vpr_argPrec /* width, or...
I    mov     width, ax
I    mov     ch, wideStage + 1
vpr_nextSwitchJmp:
I    jmp     vpr_nextSwitch
vpr_argPrec:
I    cmp     ch, precStage
I    jne     vpr_abandonJmp
I    mov     precision, ax /* the precision.
I    inc     ch
I    jmp     vpr_nextSwitch
case (_pr): /* when '.' is seen, precision */
I    cmp     ch, precStage
I    jnb     vpr_abandonJmp
I    mov     ch, precStage /* should follow
I    jmp     vpr_nextSwitch

```

```

/*      When a numeral is seen, it may be either part of a width, or
part of the precision, depending on the stage. */
case (nu):
case_nu:
/* when 0..9 seen */
l      xchg    ax, dx      /* move char back into AL */
l      sub     '0', '0'    /* turn '0'-'9' to 0-9 */
l      ctw
l      cmp     ch, wideStage /* make it a word */
l      ja      vpr_precNumeral /* is it part of a width spec? */
l      mov     ch, wideStage /* no, see if it's a precision */
l      xchg    ax, width
l      or      ax, ax
l      jl      vpr_nextSwitch /* first width digit ? */
l      shl     ax, 1        /* default width was -1 */
l      mov     dx, ax       /* *2 */
l      shl     ax, 1        /* *4 */
l      shl     ax, 1        /* *8 */
l      add     ax, dx        /* (*2 + *8) = *10 */
l      add     width, ax     /* width = (width * 10 + num) */
l      jmp     vpr_nextSwitch

vpr_precNumeral:
l      cmp     ch, precStage /* is it part of precision spec */
l      jne     vpr_abandonJmp /* no, it's just a literal */
l      xchg    ax, precision
l      or      ax, ax
l      jl      vpr_nextSwitch /* first precision digit ? */
l      shl     ax, 1        /* default precision was -1 */
l      mov     dx, ax       /* *2 */
l      shl     ax, 1        /* *4 */
l      shl     ax, 1        /* *8 */
l      add     ax, dx        /* (*2 + *8) = *10 */
l      add     precision, ax /* prec = (prec * 10 + numeral) */
l      jmp     vpr_nextSwitch

case (_lo):
l      or      W0 (flagSet), isLongBit /* 'l' was seen (long) */
l      mov     ch, ellStage
l      jmp     vpr_nextSwitch

case (_ld):
l      or      W0 (flagSet), LongDoubleBit /* 'L' was seen (long double) */
l      and     W0 (flagSet), not isLongBit
l      mov     ch, ellStage
l      jmp     vpr_nextSwitch

case (_sh):
l      and     W0 (flagSet), not isLongBit /* 'h' or 'H' was seen (short) */
#ifdef FLOATS_32_BIT
l      or      W0 (flagSet), floatBit /* 4 byte float parameter */
#endif
l      mov     ch, ellStage
l      jmp     vpr_nextSwitch

/* =====
/* The first group of cases is for the integer conversions. */
/* =====

case (_oc):
l      mov     bh, 8        /* octal */
l      jmp     short vpr_NoSign /* BH <- Base 8 */

case (_un):
l      mov     bh, 10       /* unsigned */
l      jmp     short vpr_UINT /* BH <- Base 10 */

case (_he):
l      mov     bh, 16       /* hex */
l      mov     bl, 'A' - 'X' /* BH <- Base 16 */
l      add     bl, dl        /* Adjust for aAbBcC etc later */

vpr_NoSign:
l      mov     BY0 (plusSign), 0 /* It's an unsigned operand */
/* jmp     short vpr_UINT */

vpr_UINT:
l      mov     BY0 (isSigned), 0
l      mov     fc, dl        /* 'fc' <- orig fmt char */
l      LES     di, argP

```

```

I      mov     ax, ES_ [di]          /* AX <- word arg at ES:DI */
I      xor     dx, dx                /* zero extend by default */
I      jmp     short vpr_toAscii
case ( _de):
I      mov     bh, 10                /* decimal */
I      mov     bh, 10                /* BH <- Base 10 */
vpr_INT:
I      mov     BYO (isSigned), true
I      mov     fc, di                /* 'c' <- orig fmt char */
I      LES     di, argP              /* ES:DI -> argument word[0] */
I      mov     ax, ES_ [di]          /* AX <- word arg at ES:DI */
I      cwd                     /* sign-extend by default */
vpr_toAscii:
I      inc     di                    /* ES:DI -> next arg word */
I      inc     di
I      mov     formP, si              /* remember progress through format */
I      test    W0 (flagSet), isLongBit /* short or long int ? */
I      je      vpr_shortInt          /* If the operand is a long */
I      mov     dx, ES_ [di]          /* DX:AX holds long argument */
I      inc     di                    /* ES:DI -> next arg word */
I      inc     di
vpr_shortInt:
I      mov     argP, di              /* Save arg list pointer */
I      lea     di, tempStr [1]        /* (SS) DI <- &tempStr[1] */
I      or      ax, ax                /* Is the value zero? */
I      jnz     vpr_flag_nz           /* No, */
I      or      dx, dx                /* Is the value zero? */
I      jnz     vpr_flag_nz           /* No, */
/*

```

Check for the special ANSI case of a zero value with an explicit precision of zero.

```

I      cmp     W0 (precision), 0     /* Is it the special case? */
I      jne     vpr_doLtoA            /* No, continue */
I      mov     di, aP                /* SS:DI -> somewhere in 'S' */
I      mov     cx, width              /* If he asked for nothing, */
I      jcxz    ANSI_special_end       /* then give him nothing. */
I      cmp     cx, 1                 /* Was width default? */
I      je      ANSI_special_end       /* Assume width = 1 */
I      mov     ax, W0 (flagSet)       /* fillChar = */
I      and     ax, fillZerosBit        /* (flagSet & fillZerosBit) ? */
I      jz      blankfill              /* */
I      mov     dl, '0'                /* fillChar = '0'; */
I      jmp     short fillAnother
blankfill:
I      mov     dl, ' '                /* fillChar = ' '; */
/* for (i=0; i<width; i++) PutToS( fillChar ); */
fillAnother:
I      mov     al, dl                /* AL <- fillChar */
I      call    vpr_PutToS             /* This clobbers AX */
I      loop    fillAnother            /* CX is safe though */
ANSI_special_end:
I      jmp     vpr_NEXT              /* That's the end of this field */
/*

```

"Normal" integer-output cases wind up down here somewhere.

```

vpr_flag_nz:
I      or      W0 (flagSet), notZeroBit /* flag non-zerosness */
vpr_doLtoA:
I      push    dx                    /* Push long value (AX:DX) */
I      push    ax
#ifdef LDATA
I      push    -SS
#endif
I      push    di                    /* ie. &tempStr[1] */
I      mov     al, bh                /* AL <- numeric base/radix */
I      cbw                     /* Convert it to a word */
I      push    ax                    /* Push the radix */
I      mov     al, isSigned          /* Push the isSigned flag, */
I      push    ax                    /* AH should still be 0 */

```

```

I      push    bx                      /* BL == , howCase) */
/... longtoa(number, string, radix, signflag, basecase) .../
I      call    EXTPROC (_longtoa) /* returns pointer to string */
I      push    SS
I      pop     ES                      /* ES [di] = eP == 1+tempStr */
I      mov     dx, precision           /* ES is needed in all models */
I      or      dx, dx                 /* Is precision > 0 ? */
I      jg      vpr_countActualJmp     /* Yes. */
I      jmp     vpr_testFillZeros      /* No. */
vpr_countActualJmp:
I      jmp     vpr_countActual
/* The 'p' conversion takes either a near or a far pointer and puts
it out in the usual Intel xxxxxxxx hex style. */
case (_pt):
I      mov     fc, di                 /* pointer */
I      mov     formP, si              /* remember the type character. */
I      lea     di, tempStr            /* remember progress through format */
I      LES     bx, argP
I      push    ES [bx]                /* fetch the argument.w0 */
I      inc     bx
I      inc     bx
I      mov     argP, bx
I      test    W0 (flagSet), farPtrBit
I      jz      vpr_ptrLSW
I      push    ES [bx]                /* fetch the argument.w1 */
I      inc     bx
I      inc     bx
I      mov     argP, bx
I      push    SS
I      pop     ES
I      call    Hex4
/*      add     di, 4                  Hex4 does this */
I      mov     al, '?'
I      stosb
vpr_ptrLSW:
I      push    SS
I      pop     ES
I      call    Hex4
/*      add     di, 4                  Hex4 does this */
I      mov     BY0 (SS [di]), 0
I      mov     BY0 (isSigned), 0
I      and     W0 (flagSet), NOT notZeroBit
I      lea     cx, tempStr
I      sub     di, cx
I      xchg    cx, di                 /* CX = len, DI = tempStr */
I      mov     dx, precision
I      cmp     dx, cx
I      jg      vpr_ptrEnd
I      mov     dx, cx
vpr_ptrEnd:
I      jmp     vpr_testFillZeros
/* The 'c' conversion takes a character as parameter. Note, however,
that the character occupies an (int) sized cell in the argument
list. */
case (_ch):
I      mov     formP, si              /* char */
I      mov     fc, di                 /* remember progress through format */
I      LES     di, argP               /* remember the type character */
I      mov     ax, ES [di]
I      add     W0 (argP), 2
I      push    SS
I      pop     ES
I      lea     di, tempStr [1]
I      xor     ah, ah                 /* terminate the temporary string. */
I      mov     ES [di], ax
I      mov     cx, 1
I      jmp     vpr_CopyLen
/* The 's' conversion takes a string (char *) as argument and copies

```



```

the string to the output buffer.
case (_st):
I      mov     formP, si      /* string */
I      mov     fc, di        /* remember progress through format */
I      LES     di, argP      /* remember the type character */
I      test    W0 (flagSet), farPtrBit
I      jnz     vpr_farString
#ifdef HUGE
I      jmp     vpr_abandonJmp /* DS can't be assumed in HUGE model */
#else
I      mov     di, ES [di]   /* [di] = (DS:char *) *(argP++) */
I      add     W0 (argP), 2
I      push    DS
I      pop     ES
I      or      di, di
I      jmp     short vpr_countString
#endif
vpr_farString:
I      les     di, ES [di]   /* ES: [di] = (char *) *(argP++) */
I      add     W0 (argP), 4
I      mov     ax, es
I      or      ax, di
vpr_countString:
I      jnz     NotANullPtr
I      push    DS
I      pop     ES
I      mov     di, offset NullString
NotANullPtr:
I      call    vpr_strlen    /* CX = strlen (ES: [di]) */
I      cmp     cx, precision
I      jna     vpr_CopyLenJmp
I      mov     cx, precision /* precision may truncate string. */
vpr_CopyLenJmp:
I      jmp     vpr_CopyLen
/* All real-number conversions are done by _realcvt. */
case (_fl):
I      mov     formP, si      /* float */
I      mov     fc, di        /* remember progress through format */
I      LES     di, argP      /* remember the type character */
I      mov     cx, precision
I      or      cx, cx        /* is precision defaulted ? */
I      jnl     vpr_cvtReal
I      mov     cx, 6
vpr_cvtReal:
#ifdef LDATA
I      push    ES
#endif
I      push    di            /* (valueP */
I      push    cx            /* , ndec */
#ifdef LDATA
I      push    SS
#endif
I      lea     bx, tempStr [1]
I      push    bx            /* , cP */
I      push    dx            /* , formCh */
I      mov     ax, altFormatBit
I      and     ax, W0 (flagSet)
I      push    ax            /* , altFormat */
/* Determine the argument type float/double/long double.
   Save the size of the argument type. */
I      mov     ax, W0 (flagSet)
#ifdef FLOATS_32_BIT
I      test    ax, floatBit
I      jz      try_long
I      mov     ax, F_4byteFloat
I      mov     W0 (flt_arg_len), 4
I      jmp     short push_argtype
#endif
try_long :

```

```

1      test    ax, LongDoubleBit
1      jz      its_plain_double
1      mov     ax, F_10byteFloat
1      mov     W0 (flt_arg_len), 10
1      jmp     short push_argtype
its_plain_double:
1      mov     W0 (flt_arg_len), 8
1      mov     ax, F_8byteFloat
push_argtype:
1      push    ax
1      call    EXTPROC (_realcvt)
1      mov     ax, flt_arg_len
1      add     W0 (argP), ax          /* argP += sizeof(arg) */
1      push    SS
1      pop     ES
1      lea     di, tempStr [1]        /* ES_ [di] = cP == 1+tempStr */
vpr_testFillZeros:
1      test    W0 (flagSet), fillZerosBit
1      jz      vpr_NUMERIC
1      mov     dx, width
1      or      dx, dx
1      jng     vpr_NUMERIC
vpr_countActual:
1      call    vpr_strlen              /* Get strlen of result string */
1      sub     dx, cx                 /* CX <- strlen (ES:[DI]) */
1      jng     vpr_NUMERIC            /* DX <- leading 0 count(if any) */
1      mov     leadZ, dx              /* leadZ, defaulted to 0 before */
1      mov     leadZ, dx              /* leadZ <- (width or prec-len) */
/* If arrived here, then tempStr contains the result of a numeric
conversion. It may be necessary the prefix the number with
a mandatory sign or space. */
vpr_NUMERIC:
1      mov     al, plusSign           /* ES must be well defined ! */
1      or      al, al                 /* Do we need a sign? */
1      jz      vpr_COPY               /* No, not required */
1      cmp     BY0 (ES: [di]), 0
1      je      vpr_COPY              /* No, its there already */
1      sub     W0 (leadZ), 1          /* Yes, leading 0's are 1 less */
1      adc     W0 (leadZ), 0          /* don't allow negative leadZ */
1      dec     di                     /* Back up 1 in the string */
1      mov     ES: [di], al           /* and insert the sign */
/* If arrived here then ES: [di] = cP points to the converted string,
which must now be padded, aligned, and copied to the output. */
vpr_COPY:
1      call    vpr_strlen              /* CX = strlen (ES: [di]) */
vpr_CopyLen:
1      mov     si, di                 /* comes from %c or %s section */
1      mov     di, aP                 /* cP <- ES: [si] */
1      mov     bx, width              /* BX <- width */
1      mov     ax, notZeroBit + altFormatBit
1      and     ax, W0 (flagSet)
1      cmp     ax, notZeroBit + altFormatBit
1      jne     goto_doLead
1      mov     ah, 'c'                /* AH <- original format char */
1      cmp     ah, 'o'                /* Is it alternate octal form? */
1      jne     vpr_maybeAltHex        /* No, try the next one */
1      cmp     W0 (leadZ), 0          /* Yes, alternate mode w/octal */
1      jg      goto_doLead            /* requires at least */
1      mov     W0 (leadZ), 1          /* one leading zero. */
goto_doLead:
1      jmp     vpr_doLead;
vpr_maybeAltHex:
1      cmp     ah, 'x'                /* Is it alternate hex form? */
1      je      vpr_isAltHex           /* Yes, send */
1      cmp     ah, 'X'                /* "0x" or "0X" prefix. */
1      jne     vpr_doLead            /* No, insert leading 0's */
vpr_isAltHex:
1      or      W0 (flagSet), alt0xBit /* flagSet |= alt0xBit; */
1      dec     bx                     /* width -= 2; */
1      dec     bx

```

```

I      sub    W0 (leadZ), 2          /* leadZ -= 2; */
I      jnl    vpr_doLead             /* Still leading 0's? */
I      mov    W0 (leadZ), 0          /* No more leading 0's */
vpr_doLead:
I      add    cx, leadZ              /* CX <- len + leadZ */
/* Is result to be left justified? */
I      test   W0 (flagSet), leftJustBit
I      jnz    vpr_check0x
I      jmp     short vpr_nextJust     /* (! leftJust) == leftFill */
vpr_justLoop:
I      mov     al, ' '
I      call    vpr_PutToS
I      dec     bx
vpr_nextJust:
I      cmp     bx, cx
I      jg      vpr_justLoop
vpr_check0x:
I      test    W0 (flagSet), alt0xBit /* Need alternate hex form? */
I      jz      vpr_checkLeadZ         /* No, */
I      mov     al, '0'                /* Yes, Send "0x" or "0X" */
I      call    vpr_PutToS
I      mov     al, 'c'                /* original 'c' is 'x' or 'X' */
I      call    vpr_PutToS
/* leading zero fill required ? */
vpr_checkLeadZ:
I      mov     dx, leadZ
I      or      dx, dx
I      jng     vpr_actualCopy
I      sub     cx, dx                 /* len -= leadZ */
I      sub     bx, dx                 /* width -= leadZ */
I      mov     al, ES: [si]           /* any leading sign must be */
I      cmp     al, ' '                /* copied before the */
I      je      vpr_leadSign           /* leading zeros. */
I      cmp     al, '-'
I      je      vpr_leadSign
I      cmp     al, '+'
I      jne     vpr_signedLead
vpr_leadSign:
I      lods     BY0 (ES: [si])         /* Get the 'sign' */
I      call     vpr_PutToS             /* Send the 'sign' */
I      dec     cx
I      dec     bx                     /* anticipates actualCopy */
vpr_signedLead:
I      xchg     cx, dx                 /* leading zeros follow sign */
I      jcxz     vpr_leadDone
vpr_leadZero:
I      mov     al, '0'
I      call     vpr_PutToS
I      loop     vpr_leadZero
vpr_leadDone:
I      xchg     cx, dx
/* Now we copy the actual converted string from tempStr to output. */
vpr_actualCopy:
I      jcxz     vpr_copied             /* Degenerate case? */
I      sub     bx, cx                 /* No, width -= len; */
/* this is the high-point of _vprinter ! */
vpr_copyLoop:
I      lods     BY0 (ES: [si])         /* ES:SI -> tempStr */
I      mov     BY0 (SS: [di]), al      /* SS:DI -> S */
I      inc     di
I      dec     BY0 (Scount)
I      jg      vpr_loopTest
I      call     vpr_CallPutter
vpr_loopTest:
I      loop     vpr_copyLoop
vpr_copied:
/* Is the field to be right-filled ? */
I      or      bx, bx                 /* any remaining width ? */
I      jng     vpr_done
I      mov     cx, bx
vpr_rightLoop:

```

```

1      mov     al, ' '
1      call    vpr_PutToS
1      loop    vpr_rightLoop
/* If arrive here, the conversion has been done and copied to output. */
vpr_done:
1      jmp     vpr_NEXT
case (_ns): /* number sent */
1      mov     formP, si /* remember progress through format */
1      LES     di, argP
1      test    W0 (flagSet), farPtrBit
1      jnz     vpr_farCount
#ifdef HUGE
1      jmp     vpr_abandonJump /* DS can't be assumed in HUGE model */
#else
1      mov     di, ES [di] /* [di] = (DS:char *) *(argP++) */
1      add     W0 (argP), 2
1      push    DS
1      pop     ES
1      jmp     short vpr_makeCount
#endif
vpr_farCount:
1      les     di, ES [di] /* ES: [di] = (char *) *(argP++) */
1      add     W0 (argP), 4
vpr_makeCount:
1      mov     ax, Ssize
1      sub     al, Scount
1      add     ax, totalSent
1      mov     ES: [di], ax
1      jmp     vpr_NEXTap
case (zz): /* \0 characters, unexpected end of format string */
case (dc): /* ordinary "don't care" chars in the wrong position */
case (pc): /* '%' percent characters in the wrong position */
: /* goto vpr_abandon */
) /* end switch */
/* If the format goes badly wrong, then copy it literally to the output
and abandon the conversion. */
vpr_abandon:
1      mov     si, abandonP
#ifdef LDATA
1      mov     ES, W1 (formP)
#endif
1      mov     di, aP
1      mov     al, '%'
vpr_abandLoop:
1      call    vpr_PutToS
1      lods    BY0 (ES [si])
1      or      al, al
1      jnz     vpr_abandLoop
/* If arrived here then the function has finished (either correctly or not). */
vpr_respond:
1      cmp     BY0 (Scount), Ssize /* anything waiting to be written ? */
1      jnl     vpr_end
1      call    vpr_CallPutter
vpr_end:
1      pop     ES
1      return totalSent;
}
#pragma warn use

```

vprintf 送格式化输出到stdout -- vprintf.c

用 法 int vprintf(char *form t, va_list param);

原 型 在 stdio.h

说 明 见printf

源 程 序

#include <stdio.h>

```
#include <_printf.h>
#include <_stdio.h>
int cdecl vprintf (const char *fmt, va_list ap)
{
    return __vprinter ((putnf *)__fputn, stdout, fmt, ap);
}
```

函 数 名 **vptr** - 返回指向屏幕位置的指针。 -- vram.cas

用 法 **void far * pascal __vptr(int x, int y)**

原型文件 **_video.h**

说 明 返回指向屏幕坐标(x,y)的指针。

源 程 序

```
#pragma inline
#include <_video.h>
#include <_dos.h>
void far * pascal __vptr(int x, int y)
{
    return MK_FP(_video.displayptr.u.seg,
        (_video.displayptr.u.off + (y-1)*_video.screenwidth + (x-1))*2);
}
```

函 数 名 **__vram** - 移动视屏 RAM 数据并检测雪花。 -- vram.cas

用 法 **void pascal __vram(void far *dst, void far *src, int len)**

原型文件 **_video.h**

说 明 在视屏 RAM 中从 SRC 移动 LEN 个字节的数据到 DST 并检测雪花。

源 程 序

```
void pascal __vram(void far *dst, void far *src, int len)
{
    int snow;
    snow = _video.snow;
    asm    push    ds
    asm    mov    cx, len                    /* Length value into CX */
    asm    jcxz    VRAMExit
    asm    les    di, dst                    /* Get pointers to data area */
    asm    lds    si, src
    asm    cld
    asm    cmp    si, di                    /* Setup move direction */
    asm    jae    VRAMSnowTest            /* Check for move direction */
    asm    mov    ax, cx                    /* Moving down?, then forward move ok */
    asm    dec    ax                        /* Nope. then start at other end */
    asm    shl    ax, 1
    asm    add    si, ax
    asm    add    di, ax
    asm    std
VRAMSnowTest:
    asm    cmp    word ptr snow, 0 /* Does video card snow ? */
    asm    jnz    VRAMStopSnow            /* Yes, wait for retrace */
    asm    rep    movsw                    /* Suppose to do both, do normal move */
    asm    jmp    short VRAMExit           /* All done */
VRAMStopSnow:
    asm    mov    dx, 3DAh                /* Suppose to wait, Point DX to CGA status port */
    asm    mov    ax, es                   /* See if both are in video seg */
    asm    mov    bx, ds
    asm    cmp    ax, bx
    asm    je    VIOStopSnow              /* Have to wait to and fro */
VRAMWait4HRetrace:
    asm    cli                            /* No ints during critical section */
VRAMSynchronize:
    asm    in    al, dx                    /* Get 6845 status */
    asm    ror    al, 1                    /* In horizontal retrace ? */
    asm    jc    VRAMSynchronize          /* If on, wait for cycle to end */
VRAMWaitForNext:
    asm    in    al, dx                    /* Get 6845 status */
}
```

```

asm    ror    al, 1          /* In horizontal retrace ? */
asm    jnc    VRAMWaitForNext /* No, wait for it to begin */
asm    movsw                   /* Move video ram word */
asm    sti                     /* Allow interrupts */
asm    loop   VRAMWait4HRetrace /* Next byte */
asm    jmp    short VRAMExit

VIOStopSnow:
VInWait4HRetrace:
asm    cli                     /* No ints during critical section */
VInSynchronize:
asm    in     al, dx           /* Get 6845 status */
asm    ror    al, 1           /* In horizontal retrace ? */
asm    jc     VInSynchronize  /* If on, wait for cycle to end */
VInWaitForNext:
asm    in     al, dx           /* Get 6845 status */
asm    ror    al, 1           /* In horizontal retrace ? */
asm    jnc    VInWaitForNext  /* No, wait for it to begin */
asm    lodsw                   /* Get word from video ram */
asm    sti                     /* Allow interrupts */
asm    mov    bx, ax           /* Save word */
asm    cli                     /* No ints during critical section */
VOutSynchronize:
asm    in     al, dx           /* Get 6845 status */
asm    ror    al, 1           /* In horizontal retrace ? */
asm    jc     VOutSynchronize /* If on, wait for cycle to end */
VOutWaitForNext:
asm    in     al, dx           /* Get 6845 status */
asm    ror    al, 1           /* In horizontal retrace ? */
asm    jnc    VOutWaitForNext /* No, wait for it to begin */
asm    mov    ax, bx           /* Get word to store */
asm    stosw                   /* Put word in video ram */
asm    sti                     /* Allow interrupts */
asm    loop   VInWait4HRetrace /* Next byte */
VRAMExit:
asm    cld                     /* Restore Direction Flag */
asm    pop    ds
return;
)

```

vscanf 从 stdin 中执行格式化输入

-- vscanf.c

用法 int vscanf (char *format, va_list param);

原型在 stdio.h

说明 见...scanf

源程序

```

#include <stdarg.h>
#include <stdio.h>
#include <_scanf.h>
#undef ungetc /* remove the macro version */
int cdecl vscanf (const char *fmt, va_list ap)
{
#pragma warn -sus
return _scanner (
    (int (*) (void *)) fgetc,
    (void (*) (int ch, void *)) ungetc,
    fmt,
    ap,
    stdin,
);
#pragma warn .sus
}

```

vsprintf 送格式化输出到串中

-- sprintf.c

用法 int vsprintf(char *string, char *format, va_list param);

原型在 stdio.h

说 明 见 printf

源 程 序

```
#include <stdio.h>
#include <mem.h>
#include <string.h>
#include <_printf.h>
int cdecl vsprintf(char *bufP, const char *fmt, va_list ap)
{
    *bufP = 0;
    return __vprinter ((putnF *)strputn, &bufP, fmt, ap);
}
```

vsscanf 从流中执行格式化输入 -- sscanf.c

用 法 int vsscanf (char *s, char *format, va_list param);

原 型 在 stdio.h

说 明 见 sscanf

源 程 序

```
#include <stdarg.h>
#include <stdio.h>
#include <scanf.h>
int cdecl vsscanf(const char *buf, const char *fmt, va_list ap)
{
    #pragma warn -sus
    return _scanner(
        (int (*)(void *)) Get,
        (void (*)(int ch, void *)) UnGet,
        &buf,
        fmt,
        ap
    );
    #pragma warn .sus
}
```

wherex 给出窗口内水平光标位置 -- wherexy.c

用 法 int wherex (void);

相关函数

用 法 int wherey (void);

原 型 在 conio.h

说 明 wherex 返回当前光标位置（在当前文本窗口中）的 x 坐标。wherey 返回当前位置（在当前文本窗口中）的 y 坐标。

返 回 值 wherex 返回 1 到 80 之间的一个整数。

wherey 返回 1 到 25 之间的一个整数。

可移植性 只适用于 IBM PC 及其兼容机，在 Turbo Pascal 中有相似的程序。

参 见 gotoxy

源 程 序

```
#include <video.h>
#include <conio.h>
int wherex(void)
{
    return (_wherex() - _video.windowx1 + 1);
}
```

wwherey 返回窗口内垂直光标位置 -- wherexy.c

用 法 int wherey (void);

原 型 在 conio.h

说 明 见 wherex

源 程 序

```
int wherex(void)
{
    return (_wherex() - _video.windowy1 + 1);
}
```

window 定义活动文本模式窗口 -- window.c

用 法 void window (int left, int top, int right, int bottom);

原 型 在 conio.h

说 明 window 在屏幕上定义一个文本窗口。如果坐标无效, 则对 window 的调用被忽略。

left 和 top 是窗口左上角的屏幕坐标。

right 和 bottom 是窗口右下角的屏幕坐标。

文本窗口的最小尺寸是 1 列乘 1 行, 缺省窗口为全屏幕, 坐标为:

80列模式: 1,1,80,25

40列模式: 1,1,40,25

可移植性 只适用于 IBM PC 及其兼容机, 在 Turbo Pascal 中有相应的子程序。

参 见 gettextinfo, textmode

源 程 序

```
#include < video.h>
#include < conio.h>
void window(int left, int top, int right, int bottom)
{
    left    -= 1;
    right   -= 1;
    top     -= 1;
    bottom  -= 1;
    /* consistency checking */
    if (left < 0 || right >= _video.screenwidth ||
        top < 0 || bottom >= _video.screenheight ||
        (right - left < 0) || (bottom - top < 0)) return;
    _video.windowx1 = left;
    _video.windowx2 = right;
    _video.windowy1 = top;
    _video.windowy2 = bottom;
    _DL = left;
    _DH = top;
    _AH = V_SET_CURSOR_POS;
    _BH = 0;
    _VideoInt();
}
```

_write 写到一个文件中 -- writea.cas

用 法 _write(int handle, void *buf, int nbyte);

原 型 在 io.h

说 明 见 write

源 程 序

```
#pragma inline
#include < asmrules.h>
#include < io.h>
#include < fcntl.h>
#include < io.h>
int _write(int fd, void *buf, unsigned len)
{
    if (_openfd[fd] & O_APPEND)
        lseek (fd, 0L, SEEK_END);

    pushDS
asm    mov     ah,40h
asm    mov     bx,fd
asm    mov     cx,len
```



```

asm    LDS     dx, buf
asm    int     21h
asm    popDS
asm    jc      writeFailed
asm    push    ax
asm    _openfd [fd] |= O_CHANGED;
asm    pop     ax
asm    return   _AX;
_writeFailed:
return __IOerror (_AX);
}

```

write 写到一个文件中 -- write.c

用法 `int write(int handle, void *buf, int nbyte);`

相关函数

用法 `_write (int handle, void *buf, int nbyte);`

原型在 `io.h`

说明 `write` 和 `_write` 都写数据缓冲区内容到由 `handle` 给定的文件或设备中。

`handle` 是一个从 `creat`, `open`, `dup`, `dup2` 或 `fcntl` 调用中得到的文件句柄。

这些函数从 `buf` 所指的缓冲区中写 `nbyte` 字节到与 `handle` 相关的文件中。当 `write` 用于写一个文本文件时，写到文件中的字节数不会多于所要求的字节数。

对于文本文件，当 `write` 发现一换行符(LF)时，它输出一个 CR-LF 对，`_write` 不作这种转换，因为它操作的所有文件均为二进制文件。

如果实际写的字节数少于所要求的字节数，就表示有错误存在，可能是磁盘空间满。

对于磁盘或磁盘文件，写总是从当前文件指针（见 `lseek`）开始执行；对于设备，字节直接传送到设备中。

对于用 `O_APPEND` 选择项打开的文件，`write` 在写数据以前，把文件指针指向 EOF。

返回值 两个函数均返回所写的字节数，使用 `write` 写一文本文件，不包括回车符。在出现错误时，每一函数均返回-1，并置全局变量 `errno` 为下列值之一：

EACCES	无此权限
EBADF	非法文件号

可移植性 `write` 用于 UNIX 系统

`_write` 适用于 MS-DOS

参 见 `creat`, `dup`, `lseek`, `open`

源程序

```

#include <io.h>
#include <fcntl.h>
#define SIZE 128
#pragma warn -sig
int write(int fd, void *buf, unsigned int len)
{
    register unsigned chunk;
    unsigned res;
    register char *tbuf;
    unsigned remainder;
    char c;
    char *sbuf;
    char crbuf[1 + SIZE];
    if ((len + 1) < 2)
        return (0); /* can't write 0 or 65535 bytes */
    if (_openfd[fd] & O_BINARY)
        return (_write (fd, buf, len));
    _openfd[fd] &= ~O_EOF; /* assume we go beyond ^Z EOF */
    sbuf = buf;
    remainder = len;

```


从数的开头就与指数的偏移量对应。

返回值 __xcvt 返回该数的十进制指数值。

源程序

```
#pragma inline
#include <asmrules.h>
#include <_printf.h>
#include <math.h>
#define I asm
#pragma warn -use
int pascal __xcvt(void *valP, int digits, int *signP, char *strP, int ftype)
{
    short ten = 10;
    short SW; /* INDP status word */
    char frac [10]; /* tenbyte BCD integer */
    register SI, DI; /* prevent compiler making usage */
    I push ES
    #if (! LDATA)
    I mov ax, ds
    I mov es, ax
    #endif
    I LES di, valP /* ES:DI <- pointer to value */
    I mov ax, 7FFFH /* Mask for sign zapping */
    I mov bx, ftype /* types are 2,6 or 8 */
    I mov cx, es:[bx+di] /* Get original exponent word */
    I and es:[bx+di], ax /* Zap the sign bit */
    I shr bx, 1 /* Make 'type' into 0,2 or 4 */
    I shr bx, 1 /* and do an indexed jump to */
    I shl bx, 1 /* the right load instr. */
    I jmp word ptr cs:type_table[bx]
    I type_table LABEL NEAR
    I dw F4bytes /* 4 byte 'float' */
    I dw F8bytes /* 8 byte 'double' */
    I dw F10bytes /* 10 byte 'long double' */
    I F4bytes LABEL NEAR
    I FLD FLOAT (es:[di]) /* Load 32 bit 'float' */
    I jmp short its_loaded
    I F8bytes LABEL NEAR
    I FLD DOUBLE (es:[di]) /* Load 64 bit 'double' */
    I jmp short its_loaded
    I F10bytes LABEL NEAR
    I and BY0(es:[di]), 0F0H /* Can't print em' anyway */
    I FLD LONGDOUBLE (es:[di]) /* Load 80 bit 'long double' */
    /* Take original exponent word's sign & return it to caller. */
    its_loaded:
    I xor bx, bx
    I shl cx, 1 /* CF <- sign */
    I rcl bx, 1 /* BX <- sign */
    I LES di, signP /* Store result in caller space */
    I mov ES, [di], bx
    I FXAM
    I FSTSW SW /* Get the 87 Status */
    I FWAIT
    I mov ax, SW /* Load up the status word */
    I and ah, 47H /* Mask out uninteresting stuff */
    /* Zero is the most likely 'strange' number, so it's checked first.
       Remember, signs were zapped above so we only need to look for the
       positive cases! */
    I cmp ah, 40H /* +0 */
    I je zero
    I cmp ah, 05H /* +INF */
    I je its_infinity
    I cmp ah, 01H /* +NAN */
    I je its_NAN
    goto normal;
    /******
     * Special representations for 0, Infinity and NAN values.
     * *****/
    its_NAN:
```

```

1      mov     dx, NAN_number      /* dx = NAN flag */
1      jmp     short pop_and_go
/* infinity */
1      mov     dx, INF_number      /* dx = Infinity flag */
1      jmp     short pop_and_go
/* True zero and 'rounds to zero' results wind up here */
zero:
roundToZero:
1      mov     dx, 1               /* We really want 0.0E+01 */
1      mov     al, '0'
extSize:
1      mov     cx, digits          /* fill caller's string with */
1      or      cx, cx              /* either all zeros or */
1      jc      extSized           /* all nines. */
1      neg     cx
1      inc     cx                  /* digit left of decimal point */
extSized:
1      cmp     cx, XCVDIG_         /* limit caller's buffer */
1      jbe     extLimited
1      mov     cx, XCVDIG_
extLimited:
1      cld
1      lea     di, strP            /* Fill in & NULL terminate str */
1      rep     stosb
1      xchg    al, al
1      stosb
pop_and_go:
1      FSTP    ST(0)              /* clear X from stack */
1      goto    end;
/*-----
* Normal numbers are not zero, infinite or NANs.
*
* Note: upon arrival here --
* 87 TOS contains the number to convert
*-----*/
normal:
/*** FST won't do temp-reals so we have to use FSTP ***/
1      FLD     st(0)              /* Duplicate the number */
1      FSTP    LONGDOUBLE (frac) /* Save long double form */
1      FWAIT
1      mov     ax, frac[8]         /* Get new 80 bit #'s exp word */
1      sub     ax, 3FFFh          /* Remove exponent bias */
1      mov     dx, 4D10h          /* 10000h * Log10of2, rounded. */
1      imul    dx
1      xchg    ax, bx
1      mov     ah, 4DH
1      mov     al, frac[7]
1      shl     al, 1
1      mul     ah
1      add     ax, bx
1      adc     dx, 0
1      neg     ax
1      adc     dx, 0              /* DX = estimated exponent */
/      Now we are ready to do the rounding. DX estimates the decimal digits
left of the decimal point. AX contains the requested precision. */
1      mov     ax, digits
1      or      ax, ax              /* -,0,+ = decimals, dft. digits */
1      jc      digitPlaces
/      The caller has requested (-AX) decimals following the decimal point.
1      neg     ax
1      add     ax, dx              /* AX = equivalent signif. digits */
1      jl      roundToZero        /* Ignore if it rounds to zero */
digitPlaces:
1      cmp     ax, 18
1      jg      defaultPlaces
1      mov     cx, 18
defaultPlaces:
1      mov     bx, ax              /* BX = safe copy of AX */
1      sub     ax, dx

```

```

I      mov     si, ax
I      jz      adjusted      /* 10^0 == 1, so skip the multiply/divide */
I      jnl     power10
I      neg     ax
power10:
I      push    ax
I      call    EXTPROC (pow10) /* leaves result in ST */
I      pop     ax
adjust:
I      or      si, si
I      jg      increase
I      FIDIV
I      jmp     short adjusted
increase:
I      FMUL
adjusted:
I      push    bx
I      call    EXTPROC (pow10) /* leaves result in ST */
I      pop     ax
I      FCOMP
I      /* cmp ST, ST(1), then pop */
I      FSTSW SW
I      FWAIT
I      test     BY1 (SW), 45h /* test C3, C2, C0 */
I      jz      notTooHigh /* all zero implies ST > ST(1) */
I      inc     dx /* correct the estimate of decimals */
I      inc     bx /* and size of result string */
I      cmp     bx, 18
I      ja      mustShorten
I      cmp     W0 (digits), 0 /* is format F or E ? */
I      jng     notTooLow
mustShorten:
I      FIDIV W0 (ten) /* E formats: maintain requested */
I      dec     bx /* count of digits */
I      jmp     short notTooLow
/* If arrived here the number was not too high, but may be too low. */
notTooHigh:
I      mov     ax, bx
I      dec     ax
I      push    ax
I      call    EXTPROC (pow10) /* leaves result in ST */
I      pop     ax
I      FCOMP
I      /* cmp ST, ST(1), then pop */
I      FSTSW SW
I      FWAIT
I      test     BY1 (SW), 41h /* test C3, C0 */
I      jnz     notTooLow /* either non-zero implies ST <= ST(1) */
/* Adjust upward tenfold to correct the alignment. */
I      dec     dx /* correct the estimate of decimals */
I      dec     bx /* and size of result string */
I      cmp     W0 (digits), 0 /* is format F or E ? */
I      jng     notTooLow
I      FIMUL W0 (ten) /* E formats: maintain requested */
I      inc     bx /* count of digits */
/* Now convert the number in TOS into a decimal integer of up to 18
   digits. The default rounding mode applies. */
notTooLow:
I      FRNDINT /* FBSIP does not round properly ! */
I      FBSTP frac
I      LES     di, strP /* Locate the end of string .. */
I      add     di, bx
I      push    di /* .. remember it for later $%5 .. */
I      xor     al, al /* .. and put the zero terminator there. */
I      std
I      /* fill the string in reverse order */
I      stosb
/* Locate the fraction. */
I      lea     si, frac
I      mov     cx, 4 /* CL = nibble shift, CH = round-up flag */
I      FWAIT /* wait for conversion to finish. */
I      or      bx, bx

```

```

1      jnz     nextPair
/*      Fractions which may round up to 1 are checked here as a special case.          */
1      mov     ch, SS_ [si]
1      xor     ch, 1 /* enable round-up if is 1. */
1      jmp     short maybeRoundup
/*      Note that string direction is reversed, least significant digits are
        converted first.          */
nextPair:
1      mov     al, SS_ [si] /* convert the packed BCD .. */
1      inc     si
1      mov     ah, al
1      shr     ah, cl
1      and     al, 0Fh
1      add     ax, 3030h /* '00' */ /* .. to ASCII decimals */
1      stosb
1      or      ch, al /* accumulate non-zero digits */
1      dec     bx
1      jz      maybeRoundup
1      mov     al, ah
1      stosb
1      or      ch, al /* accumulate non-zero digits */
1      dec     bx
1      jnz     nextPair
maybeRoundup:
1      pop     bx /* remember end-of-string position. */
1      and     ch, 0Fh /* were any non-zero digits seen ? */
1      jnz     append
/*      If all zeros, then we can assume the leading digit will be '1'
        due to a round-up. Increment DX to correct the estimated digits.
1      inc     dx
1      inc     bx /* also increment count of digits */
1      mov     BY0 (ES_ [di+1]), '1'
/*      The caller may want more than 18 digits. We oblige, with limits,
        by appending zeros up to the intended length.          */
append:
1      mov     cx, digits
1      or      cx, cx
1      jg      zMax
1      neg     cx /* request was for fixed decimals */
1      add     cx, dx /* so add digits to get intended size */
zMax:
1      cmp     cx, __XCVDIG__ /* assumed limit to caller's buffer */
1      jna     zLimited
1      mov     cx, __XCVDIG__
zLimited:
1      mov     BY0 (ES_ [bx]), 0 /* make sure null terminated */
1      mov     ax, bx
1      sub     ax, strP /* calculate actual digits */
1      sub     cx, ax
1      jna     end /* all digits have been delivered */
appendZloop:
1      mov     W0 (ES_ [bx]), '0' /* extend the string */
1      inc     bx
1      loop    appendZloop
end:
1      cld /* reinstate default, forwards string */
1      pop     ES
1      return _DX; /* returns decimal exponent of the number. */
}
#pragma warn .use

```

函数名 _xfclose - 关闭流。

-- xfclose.c

用法 void _xfclose(void)

说明 退出时调用以关闭所有打开的流。

源程序

```

#include <stdio.h>
void _xfclose(void)
{
    register FILE *fp;
    register int i;
    for (i = OPEN_MAX; i = streams+5; --i; fp++)
        if (fp->flags & F RDWR)
            fclose(fp);
}

```

函数名 `_xflush` - 刷新流。 -- `xflush.c`

用法 `void _xflush(void)`

说明 退出时调用以刷新所有打开的流。

源程序

```

#include <stdio.h>
void _xflush(void)
{
    register FILE *fp;
    register int i = 4;
    for (fp = streams; i; i--, fp++)
        if (fp->flags & F RDWR)
            fflush(fp);
}

```

函数名 `zapline` - 删除一行。 -- `scroll.c`

用法 `static void near pascal zapline(unsigned int *linebuffer, int x1, int x2);`

返回值 无。

源程序

```

#include <conio.h>
#include <video.h>
static void near pascal zapline(unsigned int *linebuffer, int x1, int x2)
{
    CH = video.attribute;
    CL = 0x20;
    for (; x1 <= x2; x1++)
        *linebuffer++ = _CX;
}

```