c 语言库函数大全--资料收集

分类函数,所在函数库为 ctype.h　　include “stype.h”
int isalpha(int ch) 若 ch 是字母('A'-'Z','a'-'z')返回非 0 值,否则返回 0
int isalnum(int ch) 若 ch 是字母('A'-'Z','a'-'z')或数字('0'-'9'),返回非 0 值,否则返回 0
int isascii(int ch) 若 ch 是字符(ASCII 码中的 0-127)返回非 0 值,否则返回 0
int iscntrl(int ch) 若 ch 是作废字符(0x7F)或普通控制字符(0x00-0x1F),返回非 0 值,否则返回 0
int isdigit(int ch) 若 ch 是数字('0'-'9')返回非 0 值,否则返回 0
int isgraph(int ch) 若 ch 是可打印字符(不含空格)(0x21-0x7E)返回非 0 值,否则返回 0
int islower(int ch) 若 ch 是小写字母('a'-'z')返回非 0 值,否则返回 0
int isprint(int ch) 若 ch 是可打印字符(含空格)(0x20-0x7E)返回非 0 值,否则返回 0
int ispunct(int ch) 若 ch 是标点字符(0x00-0x1F)返回非 0 值,否则返回 0
int isspace(int ch) 若 ch 是空格(' '),水平制表符('\t'),回车符('\r'), 走纸换行('\f'),垂直制表符('\v'),换行符('\n'), 返回非 0 值,否则返回 0
int isupper(int ch) 若 ch 是大写字母('A'-'Z')返回非 0 值,否则返回 0
int isxdigit(int ch) 若 ch 是 16 进制数('0'-'9','A'-'F','a'-'f')返回非 0 值, 否则返回 0
int tolower(int ch) 若 ch 是大写字母('A'-'Z')返回相应的小写字母('a'-'z')
int toupper(int ch) 若 ch 是小写字母('a'-'z')返回相应的大写字母('A'-'Z')

数学函数,所在函数库为 math.h、stdlib.h、string.h、float.h
int abs(int i) 返回整型参数 i 的绝对值
double cabs(struct complex znum) 返回复数 znum 的绝对值
double fabs(double x) 返回双精度参数 x 的绝对值
long labs(long n) 返回长整型参数 n 的绝对值
double exp(double x) 返回指数函数 ex 的值
double frexp(double value,int *eptr) 返回 value=x*2n 中 x 的值,n 存贮在 eptr 中
double ldexp(double value,int exp); 返回 value*2exp 的值
double log(double x) 返回 logex 的值
double log10(double x) 返回 log10x 的值
double pow(double x,double y) 返回 xy 的值
double pow10(int p) 返回 10p 的值
double sqrt(double x) 返回 x 的开方
double acos(double x) 返回 x 的反余弦 cos-1(x)值,x 为弧度
double asin(double x) 返回 x 的反正弦 sin-1(x)值,x 为弧度
double atan(double x) 返回 x 的反正切 tan-1(x)值,x 为弧度
double atan2(double y,double x) 返回 y/x 的反正切 tan-1(x)值,y 的 x 为弧度
double cos(double x) 返回 x 的余弦 cos(x)值,x 为弧度

double sin(double x) 返回 x 的正弦 sin(x)值,x 为弧度
double tan(double x) 返回 x 的正切 tan(x)值,x 为弧度
double cosh(double x) 返回 x 的双曲余弦 cosh(x)值,x 为弧度
double sinh(double x) 返回 x 的双曲正弦 sinh(x)值,x 为弧度
double tanh(double x) 返回 x 的双曲正切 tanh(x)值,x 为弧度
double hypot(double x,double y) 返回直角三角形斜边的长度(z), x 和 y 为直角边的长度, z2=x2+y2
double ceil(double x) 返回不小于 x 的最小整数
double floor(double x) 返回不大于 x 的最大整数
void srand(unsigned seed) 初始化随机数发生器
int rand() 产生一个随机数并返回这个数
double poly(double x,int n,double c[]) 从参数产生一个多项式
double modf(double value,double *iptr) 将双精度数 value 分解成尾数和阶
double fmod(double x,double y) 返回 x/y 的余数
double frexp(double value,int *eptr) 将双精度数 value 分成尾数和阶
double atof(char *nptr) 将字符串 nptr 转换成浮点数并返回这个浮点数
double atoi(char *nptr) 将字符串 nptr 转换成整数并返回这个整数
double atol(char *nptr) 将字符串 nptr 转换成长整数并返回这个整数
char *ecvt(double value,int ndigit,int *decpt,int *sign)
将浮点数 value 转换成字符串并返回该字符串
char *fcvt(double value,int ndigit,int *decpt,int *sign)
将浮点数 value 转换成字符串并返回该字符串
char *gcvt(double value,int ndigit,char *buf)
将数 value 转换成字符串并存于 buf 中,并返回 buf 的指针
char *ultoa(unsigned long value,char *string,int radix)
将无符号整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *ltoa(long value,char *string,int radix)
将长整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *itoa(int value,char *string,int radix)
将整数 value 转换成字符串存入 string,radix 为转换时所用基数
double atof(char *nptr) 将字符串 nptr 转换成双精度数,并返回这个数,错误返回 0
int atoi(char *nptr) 将字符串 nptr 转换成整型数, 并返回这个数,错误返回 0
long atol(char *nptr) 将字符串 nptr 转换成长整型数,并返回这个数,错误返回 0
double strtod(char *str,char **endptr)将字符串 str 转换成双精度数,并返回这个数,
long strtol(char *str,char **endptr,int base)将字符串 str 转换成长整型数, 并返回这个数,
int matherr(struct exception *e) 用户修改数学错误返回信息函数(没有必要使用)
double _matherr(_mexcep why,char *fun,double *arg1p, double *arg2p,double

retval)
用户修改数学错误返回信息函数(没有必要使用)
unsigned int _clear87() 清除浮点状态字并返回原来的浮点状态
void _fpreset() 重新初使化浮点数学程序包
unsigned int _status87() 返回浮点状态字

目录函数,所在函数库为 dir.h、dos.h
int chdir(char *path) 使指定的目录 path（如:"C:\\WPS"）变成当前的工作目录,成功返回 0
int findfirst(char *pathname,struct ffblk *ffblk,int attrib)
查找指定的文件,成功返回 0
pathname 为指定的目录名和文件名,如"C:\\WPS\\TXT"
ffblk 为指定的保存文件信息的一个结构,定义如下:

```
┌──────────────────────────┐
| struct ffblk  |
|{  |
|  char ff_reserved[21]; /*DOS 保留字*/|
|  char ff_attrib; /*文件属性*/  |
|  int ff_ftime; /*文件时间*/  |
|  int ff_fdate; /*文件日期*/  |
|  long ff_fsize; /*文件长度*/  |
|  char ff_name[13]; /*文件名*/  |
|} |
└──────────────────────────┘
```

attrib 为文件属性,由以下字符代表

```
┌──────────────┬──────────────┐
| FA_RDONLY 只读文件 | FA_LABEL 卷标号 |
| FA_HIDDEN 隐藏文件 | FA_DIREC 目录  |
| FA_SYSTEM 系统文件 | FA_ARCH 档案  |
└──────────────┴──────────────┘
```

例:

```
struct ffblk ff;
findfirst("*.wps",&ff,FA_RDONLY);
```

int findnext(struct ffblk *ffblk) 取匹配 finddirst 的文件,成功返回 0
void fumerge(char *path,char *drive,char *dir,char *name,char *ext)
此函数通过盘符 drive(C:、A:等), 路径 dir(\TC、\BC\LIB 等), 文件名 name(TC、WPS 等), 扩展名 ext(.EXE、.COM 等)组成一个文件名存与 path 中.

int fnsplit(char *path,char *drive,char *dir,char *name,char *ext)
此函数将文件名 path 分解成盘符 drive(C:、A:等), 路径 dir(\TC、\BC\LIB 等), 文件名 name(TC、WPS 等),扩展名 ext(.EXE、.COM 等),并分别存入相应的变量中.
int getcurdir(int drive,char *direc)
此函数返回指定驱动器的当前工作目录名称。成功返回 0
drive 指定的驱动器(0=当前,1=A,2=B,3=C 等)
direc 保存指定驱动器当前工作路径的变量
char *getcwd(char *buf,iint n) 此函数取当前工作目录并存入 buf 中,直到 n 个字节长为为止.错误返回 NULL
int getdisk() 取当前正在使用的驱动器,返回一个整数(0=A,1=B,2=C 等)
int setdisk(int drive) 设置要使用的驱动器 drive(0=A,1=B,2=C 等), 返回可使用驱动器总数
int mkdir(char *pathname) 建立一个新的目录 pathname,成功返回 0
int rmdir(char *pathname) 删除一个目录 pathname,成功返回 0
char *mktemp(char *template) 构造一个当前目录上没有的文件名并存于 template 中
char *searchpath(char *pathname) 利用 MSDOS 找出文件 filename 所在路径, 此函数使用 DOS 的 PATH 变量,未找到文件返回 NULL

进程函数,所在函数库为 stdlib.h、process.h
void abort() 此函数通过调用具有出口代码 3 的_exit 写一个终止信息于 stderr，并异常终止程序。无返回值

int exec…装入和运行其它程序
int execl(char *pathname,char *arg0,char *arg1,…,char *argn,NULL)
int execle(char *pathname,char *arg0,char *arg1,…, char *argn,NULL,char *envp[])
int execlp(char *pathname,char *arg0,char *arg1,…,NULL)
int execlpe(char *pathname,char *arg0,char *arg1,…,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[])

exec 函数族装入并运行程序 pathname，并将参数 arg0(arg1,arg2,argv[],envp[])传递给子程序,出错返回-1。
在 exec 函数族中,后缀 l、v、p、e 添加到 exec 后，所指定的函数将具有某种操作能力。
有后缀 p 时，函数可以利用 DOS 的 PATH 变量查找子程序文件。
l 时，函数中被传递的参数个数固定。

v 时，函数中被传递的参数个数不固定。
e 时，函数传递指定参数 envp，允许改变子进程的环境，
无后缀 e 时，子进程使用当前程序的环境。

void _exit(int status) 终止当前程序,但不清理现场
void exit(int status) 终止当前程序,关闭所有文件,写缓冲区的输出(等待输出)，并调用任何寄存器的"出口函数",无返回值

int spawn…运行子程序
int spawnl(int mode,char *pathname,char *arg0,char *arg1,…, char *argn,NULL)
int spawnle(int mode,char *pathname,char *arg0,char *arg1,…, char *argn,NULL,char *envp[])
int spawnlp(int mode,char *pathname,char *arg0,char *arg1,…, char *argn,NULL)
int spawnlpe(int mode,char *pathname,char *arg0,char *arg1,…, char *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 *pathname,char *argv[])
int spawnvpe(int mode,char *pathname,char *argv[],char *envp[])

spawn 函数族在 mode 模式下运行子程序 pathname,并将参数 arg0(arg1,arg2,argv[],envp[])传递给子程序.出错返回-1
mode 为运行模式:
mode 为 P_WAIT 表示在子程序运行完后返回本程序
P_NOWAIT 表示在子程序运行时同时运行本程序(不可用)
P_OVERLAY 表示在本程序退出后运行子程序

在 spawn 函数族中,后缀 l、v、p、e 添加到 spawn 后，所指定的函数将具有某种操作能力
有后缀 p 时，函数利用 DOS 的 PATH 查找子程序文件
l 时，函数传递的参数个数固定.
v 时，函数传递的参数个数不固定.
e 时，指定参数 envp 可以传递给子程序,允许改变子程序运行环境.
无后缀 e 时,子程序使用本程序的环境.

int system(char *command)
将 MSDOS 命令 command 传递给 DOS 执行转换子程序,函数库为 math.h、stdlib.h、ctyp

e.h、float.h

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

将浮点数 value 转换成字符串并返回该字符串

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

将浮点数 value 转换成字符串并返回该字符串

char *gcvt(double value,int ndigit,char *buf)

将数 value 转换成字符串并存于 buf 中,并返回 buf 的指针

char *ultoa(unsigned long value,char *string,int radix)

将无符号整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数

char *ltoa(long value,char *string,int radix)

将长整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数

char *itoa(int value,char *string,int radix)

将整数 value 转换成字符串存入 string,radix 为转换时所用基数

double atof(char *nptr) 将字符串 nptr 转换成双精度数,并返回这个数,错误返回 0

int atoi(char *nptr) 将字符串 nptr 转换成整型数, 并返回这个数,错误返回 0

long atol(char *nptr) 将字符串 nptr 转换成长整型数,并返回这个数,错误返回 0

double strtod(char *str,char **endptr)

将字符串 str 转换成双精度数,并返回这个数,

long strtol(char *str,char **endptr,int base)

将字符串 str 转换成长整型数, 并返回这个数,

int toascii(int c) 返回 c 相应的 ASCII

int tolower(int ch) 若 ch 是大写字母('A'-'Z')返回相应的小写字母('a'- 'z')

int _tolower(int ch) 返回 ch 相应的小写字母('a'-'z')

int toupper(int ch) 若 ch 是小写字母('a'-'z')返回相应的大写字母('A'- 'Z')

int _toupper(int ch) 返回 ch 相应的大写字母('A'-'Z')

诊断函数,所在函数库为 assert.h、math.h

void assert(int test) 一个扩展成 if 语句那样的宏，如果 test 测试失败，就显示一个信息并异常终止程序,无返回值

void perror(char *string) 本函数将显示最近一次的错误信息，格式如：字符串 string:错误信息

char *strerror(char *str) 本函数返回最近一次的错误信息,格式如: 字符串 str:错误信息

int matherr(struct exception *e)

用户修改数学错误返回信息函数(没有必要使用)

double _matherr(_mexcep why,char *fun,double *arg1p, double *arg2p,double retval)

用户修改数学错误返回信息函数(没有必要使用)

输入输出子程序，函数库为 io.h、conio.h、stat.h、dos.h、stdio.h、signal.h

int kbhit() 本函数返回最近所敲的按键
int fgetchar() 从控制台(键盘)读一个字符，显示在屏幕上
int getch() 从控制台(键盘)读一个字符，不显示在屏幕上
int putch() 向控制台(键盘)写一个字符
int getchar() 从控制台(键盘)读一个字符，显示在屏幕上
int putchar() 向控制台(键盘)写一个字符
int getche() 从控制台(键盘)读一个字符，显示在屏幕上
int ungetch(int c) 把字符 c 退回给控制台(键盘)
char *cgets(char *string) 从控制台(键盘)读入字符串存于 string 中
int scanf(char *format[,argument…])
从控制台读入一个字符串,分别对各个参数进行赋值,使用 BIOS 进行输出
int vscanf(char *format,Valist param)
从控制台读入一个字符串,分别对各个参数进行赋值,使用 BIOS 进行输出,参数从 Valist param 中取得
int cscanf(char *format[,argument…])
从控制台读入一个字符串,分别对各个参数进行赋值,直接对控制台作操作,比如显示器在显示时字符时即为直接写频方式显示
int sscanf(char *string,char *format[,argument,…])
通过字符串 string，分别对各个参数进行赋值
int vsscanf(char *string,char *format,Vlist param)
通过字符串 string,分别对各个参数进行赋值,参数从 Vlist param 中取得
int puts(char *string) 发关一个字符串 string 给控制台(显示器)，使用 BIOS 进行输出
void cputs(char *string) 发送一个字符串 string 给控制台(显示器)，直接对控制台作操作,比如显示器即为直接写频方式显示
int printf(char *format[,argument,…])
发送格式化字符串输出给控制台(显示器)，使用 BIOS 进行输出
int vprintf(char *format,Valist param)
发送格式化字符串输出给控制台(显示器)，使用 BIOS 进行输出,参数从 Valist param 中取得
int cprintf(char *format[,argument,…])
发送格式化字符串输出给控制台(显示器)，直接对控制台作操作,比如显示器即为直接写频方式显示
int vcprintf(char *format,Valist param)
发送格式化字符串输出给控制台(显示器)，直接对控制台作操作,比如显示器即为直接写频方式显示，参数从 Valist param 中取得
int sprintf(char *string,char *format[,argument,…])
将字符串 string 的内容重新写为格式化后的字符串

int vsprintf(char *string,char *format,Valist param)
将字符串 string 的内容重新写为格式化后的字符串,参数从 Valist param 中取得
int rename(char *oldname,char *newname)将文件 oldname 的名称改为 newname
int ioctl(int handle,int cmd[,int *argdx,int argcx])
本函数是用来控制输入/输出设备的，请见下表：

cmd 值	功能
0	取出设备信息
1	设置设备信息
2	把 argcx 字节读入由 argdx 所指的地址
3	在 argdx 所指的地址写 argcx 字节
4	除把 handle 当作设备号(0=当前,1=A,等)之外,均和 cmd=2 时一样
5	除把 handle 当作设备号(0=当前,1=A,等)之外,均和 cmd=3 时一样
6	取输入状态
7	取输出状态
8	测试可换性;只对于 DOS 3.x
11	置分享冲突的重算计数;只对 DOS 3.x

int (*ssignal(int sig,int(*action)())() 执行软件信号(没必要使用)
int gsignal(int sig) 执行软件信号(没必要使用)

int _open(char *pathname,int access)为读或写打开一个文件，按后按 access 来确定是读文件还是写文件,access 值见下表

access 值	意义
O_RDONLY	读文件
O_WRONLY	写文件
O_RDWR	即读也写
O_NOINHERIT	若文件没有传递给子程序,则被包含
O_DENYALL	只允许当前处理必须存取的文件
O_DENYWRITE	只允许从任何其它打开的文件读
O_DENYREAD	只允许从任何其它打开的文件写
O_DENYNONE	允许其它共享打开的文件

int open(char *pathname,int access[,int permiss])为读或写打开一个文件，按后按 access 来确定是读文件还是写文件,access 值见下表

access 值	意义
O_RDONLY	读文件
O_WRONLY	写文件
O_RDWR	即读也写
O_NDELAY	没有使用;对 UNIX 系统兼容
O_APPEND	即读也写,但每次写总是在文件尾添加
O_CREAT	若文件存在,此标志无用;若不存在,建新文件
O_TRUNC	若文件存在,则长度被截为 0,属性不变
O_EXCL	未用;对 UNIX 系统兼容
O_BINARY	此标志可显示地给出以二进制方式打开文件
O_TEXT	此标志可用于显示地给出以文本方式打开文件

permiss 为文件属性,可为以下值:
S_IWRITE 允许写 S_IREAD 允许读 S_IREAD|S_IWRITE 允许读、写
int creat(char *filename,int permiss) 建立一个新文件 filename，并设定读写性。
permiss 为文件读写性，可以为以下值
S_IWRITE 允许写 S_IREAD 允许读 S_IREAD|S_IWRITE 允许读、写
int _creat(char *filename,int attrib) 建立一个新文件 filename，并设定文件属性。
attrib 为文件属性，可以为以下值
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统
int creatnew(char *filenamt,int attrib) 建立一个新文件 filename，并设定文件属性。
attrib 为文件属性，可以为以下值
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统
int creattemp(char *filenamt,int attrib) 建立一个新文件 filename，并设定文件属性。
attrib 为文件属性，可以为以下值
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统
int read(int handle,void *buf,int nbyte) 从文件号为handle的文件中读nbyte个字符存入 buf 中
int _read(int handle,void *buf,int nbyte) 从文件号为 handle 的文件中读 nbyte 个字符存入 buf 中,直接调用 MSDOS 进行操作.
int write(int handle,void *buf,int nbyte) 将 buf 中的 nbyte 个字符写入文件号为 handle 的文件中
int _write(int handle,void *buf,int nbyte) 将 buf 中的 nbyte 个字符写入文件号为 handle 的文件中
int dup(int handle) 复制一个文件处理指针 handle,返回这个指针
int dup2(int handle,int newhandle) 复制一个文件处理指针 handle 到 newhandle

int eof(int *handle) 检查文件是否结束,结束返回 1,否则返回 0

long filelength(int handle) 返回文件长度，handle 为文件号

int setmode(int handle,unsigned mode)本函数用来设定文件号为 handle 的文件的打开方式

int getftime(int handle,struct ftime *ftime)

读取文件号为 handle 的文件的时间，并将文件时间存于 ftime 结构中，成功返回 0，ftime 结构如下：

```
┌──────────────────────────────┐
| struct ftime |
| { |
| unsigned ft_tsec:5; /*秒*/ |
| unsigned ft_min:6; /*分*/ |
| unsigned ft_hour:5; /*时*/ |
| unsigned ft_day:5; /*日*/ |
| unsigned ft_month:4;/*月*/ |
| unsigned ft_year:1; /*年-1980*/ |
| } |
└──────────────────────────────┘
```

int setftime(int handle,struct ftime *ftime) 重写文件号为 handle 的文件时间,

新时间在结构 ftime 中.成功返回 0.结构 ftime 如下:

```
┌──────────────────────────────┐
| struct ftime |
| { |
| unsigned ft_tsec:5; /*秒*/ |
| unsigned ft_min:6; /*分*/ |
| unsigned ft_hour:5; /*时*/ |
| unsigned ft_day:5; /*日*/ |
| unsigned ft_month:4;/*月*/ |
| unsigned ft_year:1; /*年-1980*/ |
| } |
└──────────────────────────────┘
```

long lseek(int handle,long offset,int fromwhere)

本函数将文件号为 handle 的文件的指针移到 fromwhere 后的第 offset 个字节处.

SEEK_SET 文件开关 SEEK_CUR 当前位置 SEEK_END 文件尾

long tell(int handle) 本函数返回文件号为 handle 的文件指针,以字节表示

int isatty(int handle)本函数用来取设备 handle 的类型

int lock(int handle,long offset,long length) 对文件共享作封锁

int unlock(int handle,long offset,long length) 打开对文件共享的封锁

int close(int handle) 关闭 handle 所表示的文件处理,handle 是从_creat、creat、creatnew、creattemp、dup、dup2、_open、open 中的一个处调用获得的文件处理成功返回 0 否则返回-1,可用于 UNIX 系统
int _close(int handle) 关闭 handle 所表示的文件处理,handle 是从_creat、creat、creatnew、creattemp、dup、dup2、_open、open 中的一个处调用获得的文件处理成功返回 0 否则返回-1,只能用于 MSDOS 系统

FILE *fopen(char *filename,char *type) 打开一个文件 filename,打开方式为 type，并返回这个文件指针，type 可为以下字符串加上后缀

type	读写性	文本/2 进制文件	建新/打开旧文件
r	读	文本	打开旧的文件
w	写	文本	建新文件
a	添加	文本	有就打开无则建新
r+	读/写	不限制	打开
w+	读/写	不限制	建新文件
a+	读/添加	不限制	有就打开无则建新

可加的后缀为 t、b。加 b 表示文件以二进制形式进行操作，t 没必要使用

例:

```
#include<stdio.h>
main()
{
 FILE *fp;
 fp=fopen("C:\\WPS\\WPS.EXE","r+b");
```

FILE *fdopen(int ahndle,char *type)
FILE *freopen(char *filename,char *type,FILE *stream)
int getc(FILE *stream) 从流 stream 中读一个字符，并返回这个字符
int putc(int ch,FILE *stream) 向流 stream 写入一个字符 ch
int getw(FILE *stream) 从流 stream 读入一个整数，错误返回 EOF
int putw(int w,FILE *stream) 向流 stream 写入一个整数
int ungetc(char c,FILE *stream) 把字符 c 退回给流 stream，下一次读进的字符将是 c
int fgetc(FILE *stream) 从流 stream 处读一个字符，并返回这个字符
int fputc(int ch,FILE *stream) 将字符 ch 写入流 stream 中
char *fgets(char *string,int n,FILE *stream)

从流 stream 中读 n 个字符存入 string 中

int fputs(char *string,FILE *stream)将字符串 string 写入流 stream 中

int fread(void *ptr,int size,int nitems,FILE *stream)

从流 stream 中读入 nitems 个长度为 size 的字符串存入 ptr 中

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

向流 stream 中写入 nitems 个长度为 size 的字符串,字符串在 ptr 中

int fscanf(FILE *stream,char *format[,argument,…])

以格式化形式从流 stream 中读入一个字符串

int vfscanf(FILE *stream,char *format,Valist param)

以格式化形式从流 stream 中读入一个字符串,参数从 Valist param 中取得

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

以格式化形式将一个字符串写给指定的流 stream

int vfprintf(FILE *stream,char *format,Valist param)

以格式化形式将一个字符串写给指定的流 stream,参数从 Valist param 中取得

int fseek(FILE *stream,long offset,int fromwhere)

函数把文件指针移到 fromwhere 所指位置的向后 offset 个字节处,fromwhere 可以为以下值:

SEEK_SET 文件开关 SEEK_CUR 当前位置 SEEK_END 文件尾

long ftell(FILE *stream) 函数返回定位在 stream 中的当前文件指针位置,以字节表示

int rewind(FILE *stream) 将当前文件指针 stream 移到文件开头

int feof(FILE *stream) 检测流 stream 上的文件指针是否在结束位置

int fileno(FILE *stream) 取流 stream 上的文件处理，并返回文件处理

int ferror(FILE *stream) 检测流 stream 上是否有读写错误，如有错误就返回 1

void clearerr(FILE *stream) 清除流 stream 上的读写错误

void setbuf(FILE *stream,char *buf) 给流 stream 指定一个缓冲区 buf

void setvbuf(FILE *stream,char *buf,int type,unsigned size)

给流 stream 指定一个缓冲区 buf,大小为 size,类型为 type,type 的值见下表

type 值	意义
_IOFBF	文件是完全缓冲区,当缓冲区是空时,下一个输入操作将企图填满整个缓冲区.在输出时,在把任何数据写到文件之前,将完全填充缓冲区.
_IOLBF	文件是行缓冲区.当缓冲区为空时,下一个输入操作将仍然企图填整个缓冲区.然而在输出时,每当新行符写到文件,缓冲区就被清洗掉.
_IONBF	文件是无缓冲的.buf 和 size 参数是被忽略的.每个输入操作将直接从文件读,每个输出操作将立即把数据写到文件中.

int fclose(FILE *stream) 关闭一个流，可以是文件或设备(例如 LPT1)
int fcloseall() 关闭所有除 stdin 或 stdout 外的流
int fflush(FILE *stream)
关闭一个流，并对缓冲区作处理处理即对读的流，将流内内容读入缓冲区；对写的流，将缓冲区内内容写入流。成功返回 0
int fflushall()
关闭所有流，并对流各自的缓冲区作处理处理即对读的流，将流内内容读入缓冲区；对写的流，将缓冲区内内容写入流。成功返回 0
int access(char *filename,int amode)
本函数检查文件 filename 并返回文件的属性，函数将属性存于 amode 中，amode 由以下位的组合构成
06 可以读、写 04 可以读 02 可以写 01 执行(忽略的) 00 文件存在
如果 filename 是一个目录,函数将只确定目录是否存在函数执行成功返回 0,否则返回-1
int chmod(char *filename,int permiss) 本函数用于设定文件 filename 的属性
permiss 可以为以下值
S_IWRITE 允许写 S_IREAD 允许读 S_IREAD|S_IWRITE 允许读、写
int _chmod(char *filename,int func[,int attrib]);
本函数用于读取或设定文件 filename 的属性，
当 func=0 时，函数返回文件的属性；当 func=1 时，函数设定文件的属性
若为设定文件属性，attrib 可以为下列常数之一
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统

接口子程序,所在函数库为:dos.h、bios.h
unsigned sleep(unsigned seconds) 暂停 seconds 微秒(百分之一秒)
int unlink(char *filename) 删除文件 filename
unsigned FP_OFF(void far *farptr) 本函数用来取远指针 farptr 的偏移量
unsigned FP_SEG(void far *farptr) 本函数用来没置远指针 farptr 的段值
void far *MK_FP(unsigned seg,unsigned off)根据段 seg 和偏移量 off 构造一个 far 指针

unsigned getpsp() 取程序段前缀的段地址,并返回这个地址
char *parsfnm(char *cmdline,struct fcb *fcbptr,int option)
函数分析一个字符串,通常,对一个文件名来说,是由 cmdline 所指的一个命令行.
文件名是放入一个 FCB 中作为一个驱动器,文件名和扩展名.FCB 是由 fcbptr 所指定的.
option 参数是 DOS 分析系统调用时,AL 文本的值.
int absread(int drive,int nsects,int sectno,void *buffer)
本函数功能为读特定的磁盘扇区,
drive 为驱动器号(0=A,1=B 等),nsects 为要读的扇区数,sectno 为开始的逻辑扇区号,buffer 为保存所读数据的保存空间

int abswrite(int drive,int nsects,int sectno,void *buffer)
本函数功能为写特定的磁盘扇区,
drive 为驱动器号(0=A,1=B 等),nsects 为要写的扇区数,sectno 为开始的逻辑扇区号,buffer 为保存所写数据的所在空间
void getdfree(int drive,struct dfree *dfreep)
本函数用来取磁盘的自由空间,
drive 为磁盘号(0=当前,1=A 等).函数将磁盘特性的由 dfreep 指向的 dfree 结构中. dfree 结构如下:

```
┌──────────────────────────────┐
| struct dfree  |
| {  |
|  unsigned df_avail; /*有用簇个数*/  |
|  unsigned df_total; /*总共簇个数*/  |
|  unsigned df_bsec; /*每个扇区字节数*/|
|  unsigned df_sclus; /*每个簇扇区数*/  |
| }  |
└──────────────────────────────┘
```

char far *getdta() 取磁盘转换地址 DTA
void setdta(char far *dta) 设置磁盘转换地址 DTA
void getfat(int drive,fatinfo *fatblkp)
本函数返回指定驱动器 drive(0=当前,1=A,2=B 等)的文件分配表信息并存入结构 fatblkp 中,结构如下:

```
┌──────────────────────────────┐
| struct fatinfo  |
| {  |
|  char fi_sclus; /*每个簇扇区数*/  |
|  char fi_fatid; /*文件分配表字节数*/|
|  int fi_nclus; /*簇的数目*/  |
|  int fi_bysec; /*每个扇区字节数*/  |
| }  |
└──────────────────────────────┘
```

void getfatd(struct fatinfo *fatblkp) 本函数返回当前驱动器的文件分配表信息, 并存入结构 fatblkp 中,结构如下:

```
┌──────────────────────────────┐
| struct fatinfo  |
| {  |
|  char fi_sclus; /*每个簇扇区数*/  |
|  char fi_fatid; /*文件分配表字节数*/|
```

```
| int fi_nclus; /*簇的数目*/ |
| int fi_bysec; /*每个扇区字节数*/ |
|} |
└──────────────────────────────┘
```

int bdos(int dosfun,unsigned dosdx,unsigned dosal)

本函数对 MSDOS 系统进行调用，dosdx 为寄存器 dx 的值,dosal 为寄存器 al 的值,dosfun 为功能号

int bdosptr(int dosfun,void *argument,unsiigned dosal)

本函数对 MSDOS 系统进行调用，

argument 为寄存器 dx 的值,dosal 为寄存器 al 的值,dosfun 为功能号

int int86(int intr_num,union REGS *inregs,union REGS *outregs)

执行 intr_num 号中断,用户定义的寄存器值存于结构 inregs 中，执行完后将返回的寄存器值存于结构 outregs 中.

int int86x(int intr_num,union REGS *inregs,union REGS *outregs, struct SREGS *segregs)

执行 intr_num 号中断,用户定义的寄存器值存于结构 inregs 中和结构 segregs 中,执行完后将返回的寄存器值存于结构 outregs 中.

int intdos(union REGS *inregs,union REGS *outregs)

本函数执行 DOS 中断 0x21 来调用一个指定的 DOS 函数,用户定义的寄存器值存于结构 inregs 中,执行完后函数将返回的寄存器值存于结构 outregs 中

int intdosx(union REGS *inregs,union REGS *outregs,struct SREGS *segregs)

本函数执行 DOS 中断 0x21 来调用一个指定的 DOS 函数,用户定义的寄存器值存于结构 inregs 和 segregs 中,执行完后函数将返回的寄存器值存于结构 outregs 中

void intr(int intr_num,struct REGPACK *preg)

本函数中一个备用的 8086 软件中断接口它能产生一个由参数 intr_num 指定的 8086 软件中断.

函数在执行软件中断前，从结构 preg 复制用户定义的各寄存器值到各个寄存器.软件中断完成后，

函数将当前各个寄存器的值复制到结构 preg 中.参数如下:

intr_num 被执行的中断号，preg 为保存用户定义的寄存器值的结构,结构如下

```
┌──────────────────────────────────────┐
|struct REGPACK |
|{ |
| unsigned r_ax,r_bx,r_cx,r_dx; |
| unsigned r_bp,r_si,r_di,r_ds,r_es,r_flags; |
|} |
└──────────────────────────────────────┘
```

函数执行完后,将新的寄存器值存于结构 preg 中

void keep(int status,int size)
以 status 状态返回 MSDOS,但程序仍保留于内存中,所占用空间由 size 决定.
void ctrlbrk(int (*fptr)()) 设置中断后的对中断的处理程序.
void disable() 禁止发生中断
void enable() 允许发生中断
void geninterrupt(int intr_num) 执行由 intr_num 所指定的软件中断
void interrupt(* getvect(int intr_num))()
返回中断号为 intr_num 的中断处理程序, 例如: old_int_10h=getvect(0x10);
void setvect(int intr_num,void interrupt(* isr)())
设置中断号为 intr_num 的中断处理程序为 isr,例如: setvect(0x10,new_int_10h);
void harderr(int (*fptr)())
定义一个硬件错误处理程序, 每当出现错误时就调用 fptr 所指的程序
void hardresume(int rescode) 硬件错误处理函数
void hardretn(int errcode) 硬件错误处理函数
int inport(int prot) 从指定的输入端口读入一个字,并返回这个字
int inportb(int port) 从指定的输入端口读入一个字节,并返回这个字节
void outport(int port,int word) 将字 word 写入指定的输出端口 port
void outportb(int port,char byte) 将字节 byte 写入指定的输出端口 port
int peek(int segment,unsigned offset)
函数返回 segment:offset 处的一个字
char peekb(int segment,unsigned offset)
函数返回 segment:offset 处的一个字节
void poke(int segment,int offset,char value)
将字 value 写到 segment:offset 处
void pokeb(int segment,int offset,int value)
将字节 value 写到 segment:offset 处
int randbrd(struct fcb *fcbptr,int reccnt)
函数利用打开 fcbptr 所指的 FCB 读 reccnt 个记录.
int randbwr(struct fcb *fcbptr,int reccnt)
函数将 fcbptr 所指的 FCB 中的 reccnt 个记录写到磁盘上
void segread(struct SREGS *segtbl)函数把段寄存器的当前值放进结构 segtbl 中
int getverify() 取检验标志的当前状态(0=检验关闭,1=检验打开)
void setverify(int value)
设置当前检验状态, value 为 0 表示关闭检验,为 1 表示打开检验
int getcbrk()本函数返回控制中断检测的当前设置
int setcbrk(int value)本函数用来设置控制中断检测为接通或断开当 value=0 时,为断开检测.当 value=1 时,为接开检测
int dosexterr(struct DOSERR *eblkp)

取扩展错误.在 DOS 出现错误后,此函数将扩充的错误信息填入 eblkp 所指的 DOSERR 结构中.
该结构定义如下:

```
struct DOSERR
{
  int exterror;/*扩展错误*/
  char class; /*错误类型*/
  char action; /*方式*/
  char locus; /*错误场所*/
} 
```

int bioscom(int cmd,char type,int port) 本函数负责对数据的通讯工作,
cmd 可以为以下值:
0 置通讯参数为字节 byte 值 1 发送字符通过通讯线输出
2 从通讯线接受字符 3 返回通讯的当前状态
port 为通讯端口,port=0 时通讯端口为 COM1,port=1 时通讯端口为 COM2,以此类推
byte 为传送或接收数据时的参数,为以下位的组合:

byte 值	意义	byte 值	意义	byte 值	意义
0x02	7 数据位	0x03	8 数据位	0x00	1 停止位
0x04	2 停止位	0x00	无奇偶性	0x08	奇数奇偶性
0x18	偶数奇偶性	0x00	110 波特	0x20	150 波特
0x40	300 波特	0x60	600 波特	0x80	1200 波特
0xA0	2400 波特	0xC0	4800 波特	0xE0	9600 波特

例如:0xE0|0x08|0x00|0x03 即表示置通讯口为 9600 波特,奇数奇偶性,1 停止位,
8 数据位. 函数返回值为一个 16 位整数,定义如下:
第 15 位 超时
第 14 位 传送移位寄存器空
第 13 位 传送固定寄存器空
第 12 位 中断检测
第 11 位 帧错误
第 10 位 奇偶错误
第 9 位 过载运行错误
第 8 位 数据就绪
第 7 位 接收线信号检测

第 6 位 环形指示器
第 5 位 数据设置就绪
第 4 位 清除发送
第 3 位 δ接收线信号检测器
第 2 位 下降边环形检测器
第 1 位 δ数据设置就绪
第 0 位 δ清除发送

int biosdisk(int cmd,int drive,int head,int track, int sector,int nsects,void *buffe r)
本函数用来对驱动器作一定的操作,cmd 为功能号, drive 为驱动器号(0=A,1=B,0x80=C,0x 81=D,0x82=E 等).cmd 可为以下值:

0 重置软磁盘系统.这强迫驱动器控制器来执行硬复位.忽略所有其它参数.
1 返回最后的硬盘操作状态.忽略所有其它参数
2 读一个或多个磁盘扇区到内存.读开始的扇区由 head、track、sector 给出。扇区号由 nsec ts 给出。把每个扇区 512 个字节的数据读入 buffer
3 从内存读数据写到一个或多个扇区。写开始的扇区由 head、track、sector 给出。扇区号由 nsects 给出。所写数据在 buffer 中，每扇区 512 个字节。
4 检验一个或多个扇区。开始扇区由 head、track、sector 给出。扇区号由 nsects 给出。
5 格式化一个磁道，该磁道由 head 和 track 给出。buffer 指向写在指定 track 上的扇区磁头器的一个表。以下 cmd 值只允许用于 XT 或 AT 微机:
6 格式化一个磁道，并置坏扇区标志。
7 格式化指定磁道上的驱动器开头。
8 返回当前驱动器参数，驱动器信息返回写在 buffer 中(以四个字节表示)。
9 初始化一对驱动器特性。
10 执行一个长的读，每个扇区读 512 加 4 个额外字节
11 执行一个长的写，每个扇区写 512 加 4 个额外字节
12 执行一个磁盘查找
13 交替磁盘复位
14 读扇区缓冲区
15 写扇区缓冲区
16 检查指定的驱动器是否就绪
17 复核驱动器
18 控制器 RAM 诊断
19 驱动器诊断
20 控制器内部诊

函数返回由下列位组合成的状态字节：
0x00 操作成功
0x01 坏的命令
0x02 地址标记找不到
0x04 记录找不到
0x05 重置失败
0x07 驱动参数活动失败
0x09 企图 DMA 经过 64K 界限
0x0B 检查坏的磁盘标记
0x10 坏的 ECC 在磁盘上读
0x11 ECC 校正的数据错误（注意它不是错误）
0x20 控制器失效
0x40 查找失败
0x80 响应的连接失败
0xBB 出现无定义错误
0xFF 读出操作失败

int biodquip()
检查设备，函数返回一字节，该字节每一位表示一个信息，如下：
第 15 位 打印机号
第 14 位 打印机号
第 13 位 未使用
第 12 位 连接游戏 I/O
第 11 位 RS232 端口号
第 8 位 未使用
第 7 位 软磁盘号
第 6 位 软磁盘号,
00 为 1 号驱动器,01 为 2 号驱动器,10 为 3 号驱动器,11 为 4 号驱动器

第 5 位 初始化
第 4 位 显示器模式
00 为未使用，01 为 40x25BW 彩色显示卡
10 为 80x25BW 彩色显示卡，11 为 80x25BW 单色显示卡
第 3 位 母扦件
第 2 位 随机存贮器容量,00 为 16K,01 为 32K,10 为 48K,11 为 64K
第 1 位 浮点共用处理器
第 0 位 从软磁盘引导

int bioskey(int cmd)本函数用来执行各种键盘操作，由 cmd 确定操作。

cmd 可为以下值：

0 返回敲键盘上的下一个键。若低 8 位为非 0,即为 ASCII 字符；若低 8 位为 0,则返回扩充了的键盘代码。

1 测试键盘是否可用于读。返回 0 表示没有键可用；否则返回下一次敲键之值。敲键本身一直保持由下次调用具的 cmd 值为 0 的 bioskey 所返回的值。

2 返回当前的键盘状态，由返回整数的每一个位表示，见下表：

位	为 0 时意义	为 1 时意义
7	插入状态	改写状态
6	大写状态	小写状态
5	数字状态，NumLock 灯亮	光标状态，NumLock 灯熄
4	ScrollLock 灯亮	ScrollLock 灯熄
3	Alt 按下	Alt 未按下
2	Ctrl 按下	Ctrl 未按下
1	左 Shift 按下	左 Shift 未按下
0	右 Shift 按下	右 Shift 未按下

int biosmemory() 返回内存大小,以 K 为单位.

int biosprint(int cmd,int byte,int port) 控制打印机的输入/输出.

port 为打印机号,0 为 LPT1,1 为 LPT2,2 为 LPT3 等

cmd 可以为以下值:

0 打印字符,将字符 byte 送到打印机

1 打印机端口初始化

2 读打印机状态

函数返回值由以下位值组成表示当前打印机状态

0x01 设备时间超时

0x08 输入/输出错误

0x10 选择的

0x20 走纸

0x40 认可

0x80 不忙碌

int biostime(int cmd,long newtime)计时器控制,cmd 为功能号,可为以下值

0 函数返回计时器的当前值

1 将计时器设为新值 newtime

struct country *country(int countrycmode,struct country *countryp)
本函数用来控制某一国家的相关信息,如日期,时间,货币等.
若 countryp=-1 时,当前的国家置为 countrycode 值(必须为非 0).否则,由 countryp 所指向的 country 结构用下列的国家相关信息填充:
(1)当前的国家(若 countrycode 为 0 或 2)由 countrycode 所给定的国家.
结构 country 定义如下:

```
struct country
{
  int co_date; /*日期格式*/
  char co_curr[5]; /*货币符号*/
  char co_thsep[2]; /*数字分隔符*/
  char co_desep[2]; /*小数点*/
  char co_dtsep[2]; /*日期分隔符*/
  char co_tmsep[2]; /*时间分隔符*/
  char co_currstyle; /*货币形式*/
  char co_digits; /*有效数字*/
  int (far *co_case)(); /*事件处理函数*/
  char co_dasep; /*数据分隔符*/
  char co_fill[10]; /*补充字符*/
}
```

co_date 的值所代表的日期格式是:
0 月日年 1 日月年 2 年月日
co_currstrle 的值所代表的货币显示方式是
0 货币符号在数值前,中间无空格
1 货币符号在数值后,中间无空格
2 货币符号在数值前,中间有空格
3 货币符号在数值后,中间有空格

操作函数,所在函数库为 string.h、mem.h
mem…操作存贮数组
void *memccpy(void *destin,void *source,unsigned char ch,unsigned n)
void *memchr(void *s,char ch,unsigned n)
void *memcmp(void *s1,void *s2,unsigned n)
int memicmp(void *s1,void *s2,unsigned n)
void *memmove(void *destin,void *source,unsigned n)

void *memcpy(void *destin,void *source,unsigned n)
void *memset(void *s,char ch,unsigned n)

这些函数,mem…系列的所有成员均操作存贮数组.在所有这些函数中,数组是 n 字节长.
memcpy 从 source 复制一个 n 字节的块到 destin.如果源块和目标块重迭,则选择复制方向,以例正确地复制覆盖的字节.
memmove 与 memcpy 相同. memset 将 s 的所有字节置于字节 ch 中.s 数组的长度由 n 给出.
memcmp 比较正好是 n 字节长的两个字符串 s1 和 s2.些函数按无符号字符比较字节,因此,memcmp("0xFF","\x7F",1)返回值大于 0. memicmp 比较 s1 和 s2 的前 n 个字节,不管字符大写或小写.
memccpy 从 source 复制字节到 destin.复制一结束就发生下列任一情况:
(1)字符 ch 首选复制到 destin.
(2)n 个字节已复制到 destin.
memchr 对字符 ch 检索 s 数组的前 n 个字节.
返回值:memmove 和 memcpy 返回 destin
memset 返回 s 的值
memcmp 和 memicmp——┬——若 s1<s2 返回值小于 0
├——若 s1=s2 返回值等于 0
└——若 s1>s2 返回值大于 0
memccpy 若复制了 ch,则返回直接跟随 ch 的在 destin 中的字节的一个指针;

否则返回 NULL
memchr 返回在 s 中首先出现 ch 的一个指针;如果在 s 数组中不出现 ch,就返回 NULL.

void movedata(int segsrc,int offsrc, int segdest,int offdest, unsigned numbytes)

本函数将源地址(segsrc:offsrc)处的 numbytes 个字节复制到目标地址(segdest:offdest)
void movemem(void *source,void *destin,unsigned len)
本函数从 source 处复制一块长 len 字节的数据到 destin.若源地址和目标地址字符串重迭,则选择复制方向,以便正确的复制数据.
void setmem(void *addr,int len,char value)
本函数把 addr 所指的块的第一个字节置于字节 value 中.

str…字符串操作函数
char stpcpy(char *dest,const char *src) 将字符串 src 复制到 dest
char strcat(char *dest,const char *src) 将字符串 src 添加到 dest 末尾
char strchr(const char *s,int c) 检索并返回字符 c 在字符串 s 中第一次出现的位置

int strcmp(const char *s1,const char *s2) 比较字符串 s1 与 s2 的大小,并返回 s1-s2

char strcpy(char *dest,const char *src) 将字符串 src 复制到 dest

size_t strcspn(const char *s1,const char *s2) 扫描 s1,返回在 s1 中有,在 s2 中也有的字符个数

char strdup(const char *s) 将字符串 s 复制到最近建立的单元

int stricmp(const char *s1,const char *s2) 比较字符串 s1 和 s2,并返回 s1-s2

size_t strlen(const char *s) 返回字符串 s 的长度

char strlwr(char *s)

将字符串 s 中的大写字母全部转换成小写字母,并返回转换后的字符串

char strncat(char *dest,const char *src,size_t maxlen)

将字符串 src 中最多 maxlen 个字符复制到字符串 dest 中

int strncmp(const char *s1,const char *s2,size_t maxlen)

比较字符串 s1 与 s2 中的前 maxlen 个字符

char strncpy(char *dest,const char *src,size_t maxlen)

复制 src 中的前 maxlen 个字符到 dest 中

int strnicmp(const char *s1,const char *s2,size_t maxlen)

比较字符串 s1 与 s2 中的前 maxlen 个字符

char strnset(char *s,int ch,size_t n)

将字符串 s 的前 n 个字符置于 ch 中

char strpbrk(const char *s1,const char *s2)

扫描字符串 s1,并返回在 s1 和 s2 中均有的字符个数

char strrchr(const char *s,int c)

扫描最后出现一个给定字符 c 的一个字符串 s

char strrev(char *s)

将字符串 s 中的字符全部颠倒顺序重新排列,并返回排列后的字符串

char strset(char *s,int ch)

将一个字符串 s 中的所有字符置于一个给定的字符 ch

size_t strspn(const char *s1,const char *s2)

扫描字符串 s1,并返回在 s1 和 s2 中均有的字符个数

char strstr(const char *s1,const char *s2)

扫描字符串 s2,并返回第一次出现 s1 的位置

char strtok(char *s1,const char *s2)

检索字符串 s1,该字符串 s1 是由字符串 s2 中定义的定界符所分隔

char strupr(char *s)

将字符串 s 中的小写字母全部转换成大写字母,并返回转换后的字符串

存贮分配子程序,所在函数库为 dos.h、alloc.h、malloc.h、stdlib.h、process.h

int allocmem(unsigned size,unsigned *seg)

利用 DOS 分配空闲的内存, size 为分配内存大小,seg 为分配后的内存指针

int freemem(unsigned seg)

释放先前由 allocmem 分配的内存,seg 为指定的内存指针

int setblock(int seg,int newsize)

本函数用来修改所分配的内存长度, seg 为已分配内存的内存指针,newsize 为新的长度

int brk(void *endds)

本函数用来改变分配给调用程序的数据段的空间数量,新的空间结束地址为 endds

char *sbrk(int incr)

本函数用来增加分配给调用程序的数据段的空间数量,增加 incr 个字节的空间

unsigned long coreleft() 本函数返回未用的存储区的长度,以字节为单位

void *calloc(unsigned nelem,unsigned elsize)

分配 nelem 个长度为 elsize 的内存空间并返回所分配内存的指针

void *malloc(unsigned size) 分配 size 个字节的内存空间,并返回所分配内存的指针

void free(void *ptr) 释放先前所分配的内存,所要释放的内存的指针为 ptr

void *realloc(void *ptr,unsigned newsize)

改变已分配内存的大小,ptr 为已分配有内存区域的指针,newsize 为新的长度,返回分配好的内存指针.

long farcoreleft() 本函数返回远堆中未用的存储区的长度,以字节为单位

void far *farcalloc(unsigned long units,unsigned long unitsz)

从远堆分配 units 个长度为 unitsz 的内存空间,并返回所分配内存的指针

void *farmalloc(unsigned long size)

分配 size 个字节的内存空间, 并返回分配的内存指针

void farfree(void far *block)

释放先前从远堆分配的内存空间, 所要释放的远堆内存的指针为 block

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

改变已分配的远堆内存的大小,block 为已分配有内存区域的指针,newzie 为新的长度,返回分配好的内存指针

时间日期函数,函数库为 time.h、dos.h

在时间日期函数里,主要用到的结构有以下几个:

总时间日期贮存结构 tm

```
┌──────────────────────────────────────┐
| struct tm  |
|{  |
|  int tm_sec; /*秒,0-59*/  |
|  int tm_min; /*分,0-59*/  |
|  int tm_hour; /*时,0-23*/  |
|  int tm_mday; /*天数,1-31*/  |
```

```
| int tm_mon; /*月数,0-11*/ |
| int tm_year; /*自 1900 的年数*/ |
| int tm_wday; /*自星期日的天数 0-6*/ |
| int tm_yday; /*自 1 月 1 日起的天数,0-365*/ |
| int tm_isdst; /*是否采用夏时制,采用为正数*/|
|} |
└──────────────────────────────────────┘
```

日期贮存结构 date

```
┌──────────────────────────┐
|struct date |
|{ |
| int da_year; /*自 1900 的年数*/|
| char da_day; /*天数*/ |
| char da_mon; /*月数 1=Jan*/ |
|} |
└──────────────────────────┘
```

时间贮存结构 time

```
┌──────────────────────────┐
|struct time |
|{ |
| unsigned char ti_min; /*分钟*/|
| unsigned char ti_hour; /*小时*/|
| unsigned char ti_hund; |
| unsigned char ti_sec; /*秒*/ |
| |
└──────────────────────────┘
```

char *ctime(long *clock)

本函数把 clock 所指的时间(如由函数 time 返回的时间)转换成下列格式的字符串:

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

char asctime(struct tm *tm)

本函数把指定的 tm 结构类的时间转换成下列格式的字符串:

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

double difftime(time_t time2,time_t time1)

计算结构 time2 和 time1 之间的时间差距(以秒为单位)

struct tm *gmtime(long *clock)

本函数把 clock 所指的时间(如由函数 time 返回的时间)转换成格林威治时间,并以 tm 结构形式返回

struct tm *localtime(long *clock)

本函数把 clock 所指的时间(如函数 time 返回的时间)转换成当地标准时间,并以 tm 结构形式返回

void tzset()本函数提供了对 UNIX 操作系统的兼容性

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

本函数将 dateptr 所指的日期,timeptr 所指的时间转换成 UNIX 格式, 并返回自格林威治时间 1970 年 1 月 1 日凌晨起到现在的秒数

void unixtodos(long utime,struct date *dateptr,struct time *timeptr)

本函数将自格林威治时间 1970 年 1 月 1 日凌晨起到现在的秒数 utime 转换成 DOS 格式并保存于用户所指的结构 dateptr 和 timeptr 中

void getdate(struct date *dateblk)

本函数将计算机内的日期写入结构 dateblk 中以供用户使用

void setdate(struct date *dateblk)

本函数将计算机内的日期改成由结构 dateblk 所指定的日期

void gettime(struct time *timep)

本函数将计算机内的时间写入结构 timep 中, 以供用户使用

void settime(struct time *timep)

本函数将计算机内的时间改为由结构 timep 所指的时间

long time(long *tloc)

本函数给出自格林威治时间 1970 年 1 月 1 日凌晨至现在所经过的秒数,并将该值存于 tloc 所指的单元中.

int stime(long *tp)本函数将 tp 所指的时间(例如由 time 所返回的时间)写入计算机中.