c语言函数库

目录

[第一章（C标准库） 4](#_Toc339983865)

[1. <assert.h>：诊断 4](#_Toc339983866)

[2. <ctype.h>：字符类别测试 5](#_Toc339983867)

[3. <errno.h>：错误处理 5](#_Toc339983868)

[4. <limits.h>：整型常量 6](#_Toc339983869)

[5. <locale.h>：地域环境 6](#_Toc339983870)

[6. <math.h>：数学函数 7](#_Toc339983871)

[7. <setjmp.h>：非局部跳转 8](#_Toc339983872)

[8. <signal.h>：信号 9](#_Toc339983873)

[9. <stdarg.h>：可变参数表 11](#_Toc339983874)

[10. <stddef.h>：公共定义 11](#_Toc339983875)

[11. <stdio.h>：输入输出 12](#_Toc339983876)

[12. <stdlib.h>：实用函数 13](#_Toc339983877)

[13. <time.h>：日期与时间函数 13](#_Toc339983878)

[第二章（IO函数） 14](#_Toc339983879)

[clearerr：复位错误标志函数 15](#_Toc339983880)

[feof：检测文件结束符函数 16](#_Toc339983881)

[ferror：检测流上的错误函数 17](#_Toc339983882)

[fflush：清除文件缓冲区函数 18](#_Toc339983883)

[fgetc：从流中读取字符函数 19](#_Toc339983884)

[fgetpos：取得当前文件的句柄函数 20](#_Toc339983885)

[fgets：从流中读取字符串函数 21](#_Toc339983886)

[fopen、fclose：文件的打开与关闭函数 22](#_Toc339983887)

[fprintf：格式化输出函数 23](#_Toc339983888)

[fputc：向流中输出字符函数 25](#_Toc339983889)

[fputs：向流中输出字符串函数 25](#_Toc339983890)

[fread：从流中读取字符串函数 26](#_Toc339983891)

[freopen：替换文件中数据流函数 27](#_Toc339983892)

[fscanf：格式化输入函数 28](#_Toc339983893)

[fseek：文件指针定位函数 29](#_Toc339983894)

[fsetpos：定位流上的文件指针函数 30](#_Toc339983895)

[ftell：返回当前文件指针位置函数 31](#_Toc339983896)

[fwrite：向文件写入数据函数 31](#_Toc339983897)

[getc：从流中读取字符函数 32](#_Toc339983898)

[getchar：从标准输入文件中读取字符函数 33](#_Toc339983899)

[gets：从标准输入文件中读取字符串函数 34](#_Toc339983900)

[perror：打印系统错误信息函数 35](#_Toc339983901)

[printf：产生格式化输出的函数 35](#_Toc339983902)

[putc：向指定流中输出字符函数 36](#_Toc339983903)

[putchar：向标准输出文件上输出字符 37](#_Toc339983904)

[puts：将字符串输出到终端函数 37](#_Toc339983905)

[remove：删除文件函数 38](#_Toc339983906)

[rename：重命名文件函数 39](#_Toc339983907)

[rewind：重置文件指针函数 39](#_Toc339983908)

[scanf：格式化输入函数 40](#_Toc339983909)

[setbuf、setvbuf：指定文件流的缓冲区函数 41](#_Toc339983910)

[sprintf：向字符串写入格式化数据函数 42](#_Toc339983911)

[sscanf：从缓冲区中读格式化字符串函数 43](#_Toc339983912)

[tmpfile：创建临时文件函数 44](#_Toc339983913)

[tmpnam：创建临时文件名函数 44](#_Toc339983914)

[ungetc：把字符退回到输入流函数 45](#_Toc339983915)

[第三章（字符处理函数） 47](#_Toc339983916)

[isalnum：检查字符是否是字母或数字 47](#_Toc339983917)

[isalpha：检查字符是否是字母 48](#_Toc339983918)

[isascii：检查字符是否是ASCII码 49](#_Toc339983919)

[iscntrl：检查字符是否是控制字符 49](#_Toc339983920)

[isdigit：检查字符是否是数字字符 50](#_Toc339983921)

[isgraph：检查字符是否是可打印字符（不含空格） 51](#_Toc339983922)

[islower：检查字符是否是小写字母 51](#_Toc339983923)

[isprint：检查字符是否是可打印字符（含空格） 52](#_Toc339983924)

[ispunct：检查字符是否是标点字符 53](#_Toc339983925)

[isspace：检查字符是否是空格符 53](#_Toc339983926)

[isupper：检查字符是否是大写字母 54](#_Toc339983927)

[isxdigit：检查字符是否是十六进制数字字符 55](#_Toc339983928)

[toascii：将字符转换为ASCII码 55](#_Toc339983929)

[tolower：将大写字母转换为小写字母 56](#_Toc339983930)

[toupper：将小写字母转换为大写字母 57](#_Toc339983931)

[第四章（字符串函数） 57](#_Toc339983932)

[atof：字符串转浮点型函数 58](#_Toc339983933)

[atoi：字符串转整型函数 59](#_Toc339983934)

[atol：字符串转长整型函数 60](#_Toc339983935)

[memchr：字符搜索函数 60](#_Toc339983936)

[memcmp：字符串比较函数 61](#_Toc339983937)

[memcpy： 字符串拷贝函数 62](#_Toc339983938)

[memmove： 字块移动函数 63](#_Toc339983939)

[memset：字符加载函数 64](#_Toc339983940)

[strcat：字符串连接函数 65](#_Toc339983941)

[strchr：字符串中字符首次匹配函数 66](#_Toc339983942)

[strcmp：字符串比较函数 67](#_Toc339983943)

[strcpy：字符串拷贝函数 68](#_Toc339983944)

[strcspn：字符集逆匹配函数 69](#_Toc339983945)

[strdup：字符串新建拷贝函数 70](#_Toc339983946)

[strerror：字符串错误信息函数 71](#_Toc339983947)

[strlen：计算字符串长度函数 72](#_Toc339983948)

[strlwr：字符串小写转换函数 72](#_Toc339983949)

[strncat：字符串连接函数 73](#_Toc339983950)

[strncmp：字符串子串比较函数 74](#_Toc339983951)

[strncpy：字符串子串拷贝函数 75](#_Toc339983952)

[strpbrk：字符集字符匹配函数 76](#_Toc339983953)

[strrchr：字符串中字符末次匹配函数 77](#_Toc339983954)

[strrev：字符串倒转函数 78](#_Toc339983955)

[strset：字符串设定函数 79](#_Toc339983956)

[strspn：字符集匹配函数 79](#_Toc339983957)

[strstr：字符串匹配函数 80](#_Toc339983958)

[strtod：字符串转换成双精度函数 81](#_Toc339983959)

[strtok：字符串分隔函数 82](#_Toc339983960)

[strtol：字符串转换成长整型函数 83](#_Toc339983961)

[strtoul：字符串转换成无符号长整型函数 85](#_Toc339983962)

[strupr：字符串大写转换函数 86](#_Toc339983963)

[strupr：字符串大写转换函数 86](#_Toc339983964)

[第五章（数学函数） 87](#_Toc339983965)

[abs、labs、fabs：求绝对值函数 88](#_Toc339983966)

[acos： 反余弦函数 89](#_Toc339983967)

[asin： 反正弦函数 89](#_Toc339983968)

[atan：反正切函数 90](#_Toc339983969)

[atan2：反正切函数2 90](#_Toc339983970)

[ceil：向上舍入函数 91](#_Toc339983971)

[cos ：余弦函数 91](#_Toc339983972)

[cosh： 双曲余弦函数 92](#_Toc339983973)

[div、ldiv：除法函数 92](#_Toc339983974)

[exp：求e的x次幂函数 93](#_Toc339983975)

[floor：向下舍入函数 94](#_Toc339983976)

[fmod：求模函数 95](#_Toc339983977)

[frexp：分解浮点数函数 95](#_Toc339983978)

[hypot：求直角三角形斜边长函数 96](#_Toc339983979)

[ldexp：装载浮点数函数 96](#_Toc339983980)

[log、log10：对数函数 97](#_Toc339983981)

[modf：分解双精度数函数 98](#_Toc339983982)

[pow、pow10：指数函数 98](#_Toc339983983)

[rand：产生随机整数函数 99](#_Toc339983984)

[sin：正弦函数 99](#_Toc339983985)

[sinh： 双曲正弦函数 100](#_Toc339983986)

[sqrt：开平方函数 100](#_Toc339983987)

[srand：设置随机时间的种子函数 101](#_Toc339983988)

[tan： 正切函数 102](#_Toc339983989)

[tanh：双曲正切函数 102](#_Toc339983990)

[第六章（时间和日期函数） 103](#_Toc339983991)

[asctime：日期和时间转换函数 103](#_Toc339983992)

[clock：测定运行时间函数 104](#_Toc339983993)

[ctime：时间转换函数 105](#_Toc339983994)

[difftime：计算时间差函数 105](#_Toc339983995)

[gmtime： 将日历时间转换为GMT 106](#_Toc339983996)

[localtime：把日期和时间转变为结构 107](#_Toc339983997)

[mktime：时间类型转换函数 107](#_Toc339983998)

[time：获取系统时间函数 109](#_Toc339983999)

[第七章（其它函数） 109](#_Toc339984000)

[abort：异常终止进程函数 110](#_Toc339984001)

[atexit：注册终止函数 110](#_Toc339984002)

[bsearch：二分搜索函数 111](#_Toc339984003)

[calloc：分配主存储器函数 113](#_Toc339984004)

[exit：正常终止进程函数 114](#_Toc339984005)

[free：释放内存函数 114](#_Toc339984006)

[getenv：获取环境变量 115](#_Toc339984007)

[malloc：动态分配内存函数 116](#_Toc339984008)

[qsort：快速排序函数 116](#_Toc339984009)

[realloc：重新分配主存函数 118](#_Toc339984010)

## 第一章（C标准库）

1. <assert.h>：诊断

2. <ctype.h>：字符类别测试

3. <errno.h>：错误处理

4. <limits.h>：整型常量

5. <locale.h>：地域环境

6. <math.h>：数学函数

7. <setjmp.h>：非局部跳转

8. <signal.h>：信号

9. <stdarg.h>：可变参数表

10. <stddef.h>：公共定义

11. <stdio.h>：输入输出

12. <stdlib.h>：实用函数

13. <time.h>：日期与时间函数

## 1. <assert.h>：诊断

　　<assert.h>中只定义了一个带参的宏assert，其定义形式如下：

void assert (int 表达式)

　　assert宏用于为程序增加诊断功能，它可以测试一个条件并可能使程序终止。在执行语句：

assert(表达式);

时，如果表达式为0，则在终端显示一条信息：

Assertion failed: 0, file 源文件名, line 行号

Abnormal program termination

然后调用abort终止程序的执行。

在<assert.h>中，带参宏assert是被定义为条件编译的，如果在源文件中定义了宏NDEBUG，则即使包含了头文件<assert.h>，assert宏也将被忽略。

## 2. <ctype.h>：字符类别测试

　　在头文件<ctype.h>中定义了一些测试字符的函数。在这些函数中，每个函数的参数都是整型int，而每个参数的值或者为EOF，或者为char类型的字符。<ctype.h>中定义的标准函数列表如下：

<ctype.h>中定义的函数

函数定义, 函数功能简介

int isalnum(int c), 检查字符是否是字母或数字

int isalpha(int c), 检查字符是否是字母

int isascii(int c), 检查字符是否是ASCII码

int iscntrl(int c), 检查字符是否是控制字符

int isdigit(int c), 检查字符是否是数字字符

int isgraph(int c), 检查字符是否是可打印字符

int islower(int c), 检查字符是否是小写字母

int isprint(int c), 检查字符是否是可打印字符

int ispunct(int c), 检查字符是否是标点字符

int isspace(int c), 检查字符是否是空格符

int isupper(int c), 检查字符是否是大写字母

int isxdigit(int c), 检查字符是否是十六进制数字字符

int toupper(int c), 将小写字母转换为大写字母

int tolower(int c), 将大写字母转换为小写字母

, ,

　　有关<ctype.h>中定义的这些标准函数以及一些常用的非标准字符处理函数将在第十一章中进行详细地介绍。

## 3. <errno.h>：错误处理

　　<errno.h>中定义了两个常量，一个变量。

1、 EDOM

它表示数学领域错误的错误代码。

2、 ERANGE

它表示结果超出范围的错误代码。

3、 errno

这是一个变量，该值被设置成用来指出系统调用的错误类型。

## 4. <limits.h>：整型常量

　　在头文件<limits.h>中定义了一些表示整型大小的常量。下面给出这些常量的字符表示以及含义，见下表。

<limits.h>中定义的字符常量字符常量, 取值, 含义

CHAR\_BIT, 8, char类型的位数

CHAR\_MAX, 255或127, char类型最大值

CHAR\_MIN, 0或-127, char类型最小值

INT\_MIN, -32767, int类型最小值

INT\_MAX , 32767, int类型最大值

LONG\_MAX, 2147483647, long类型最大值

LONG\_MIN, -2147483647, long类型最小值

SCHAR\_MAX, 127, signed char 类型最大值

SCHAR\_MIN, -127, signed char 类型最小值

SHRT\_MAX, 32767, short类型的最大值

SHRT\_MIN, -32767, short类型的最小值

UCHAR\_MAX, 255, unsigned char 类型最大值

UINT\_MAX, 65535, unsigned int 类型最大值

ULONG\_MAX, 4294967295, unsigned long 类型最大值

USHRT\_MAX, 65535, unsigned short类型的最大值

## 5. <locale.h>：地域环境

在<locale.h>中，定义了7个常量，一个结构，2个函数。

1、常量的定义

　　LC\_ALL：传递给setlocale的第一个参数，指定要更改该locale的哪个方面。

　　LC\_COLLATE：strcoll和strxfrm的行为。

　　LC\_CTYPE：字符处理函数。

　　LC\_MONETARY：localeconv返回的货币信息。

　　LC\_NUMERIC：localeconv返回的小数点和货币信息。

　　LC\_TIME：strftime的行为。

　　以上扩展成具有唯一取值的整型常数表达式，可作为setlocale的第一个参数。

　　NULL：由实现环境定义的空指针。

　　2、struct lconv结构

　　该结构用于存储和表示当前locale的设置。其结构定义如下：

struct lconv

{

char \*decimal\_point;

char \*thousands\_sep;

char \*grouping;

char \*int\_curr\_symbol;

char \*currency\_symbol;

char \*mon\_decimal\_point;

char \*mon\_thousands\_sep;

char \*mon\_grouping;

char \*positive\_sign;

char \*negative\_sign;

char int\_frac\_digits;

char frac\_digits;

char p\_cs\_precedes;

char p\_sep\_by\_space;

char n\_cs\_precedes;

char n\_sep\_by\_space;

char p\_sign\_posn;

char n\_sign\_posn;

};

　　3、函数

　　struct Iconv \*localeconv(void);

　　函数localeconv将一个struct Iconv类型的对象的数据成员设置成为按照当前地域环境的有关规则进行数量格式化后的相应值。

　　char \*setlocale(int category,char \* locale);

　　函数setlocale用于更改和查询程序的整个当前地域环境或部分设置。地域环境变量由参数category(上面定义的6个常量)和locale指定。

## 6. <math.h>：数学函数

　　在<math.h>中定义了一些数学函数和宏，用来实现不同种类的数学运算。下面给出<math.h>中标准数学函数的函数定义及功能简介，见下表。

<math.h>中定义的函数

函数定义, 函数功能简介

double exp(double x);, 指数运算函数, 求e的x次幂函数

double log(double x), 对数函数ln(x)

double log10(double x); , 对数函数log

double pow(double x, double y);, 指数函数(x的y次方)

double sqrt(double x); , 计算平方根函数

double ceil(double x); , 向上舍入函数

double floor(double x);, 向下舍入函数

double fabs(double x);, 求浮点数的绝对值

double ldexp(double x, int n); , 装载浮点数函数

double frexp(double x, int\* exp); , 分解浮点数函数

double modf(double x, double\* ip); , 分解双精度数函数

double fmod(double x, double y);, 求模函数

double sin(double x); , 计算x的正弦值函数

double cos(double x);, 计算x的余弦值函数

double tan(double x);, 计算x的正切值函数

double asin(double x);, 计算x的反正弦函数

double acos(double x); , 计算x的反余弦函数

double atan(double x); , 反正切函数1

double atan2(double y, double x); , 反正切函数2

double sinh(double x); , 计算x的双曲正弦值

double cosh(double x); , 计算x的双曲余弦值

double tanh(double x); , 计算x的双曲正切值

　　在标准库中，还有一些与数学计算有关的函数定义在其他头文件中。

## 7. <setjmp.h>：非局部跳转

　　在头文件<setjmp.h>中定义了一种特别的函数调用和函数返回顺序的方式。这种方式不同于以往的函数调用和返回顺序，它允许程序流程立即从一个深层嵌套的函数中返回。

　　<setjmp.h>中定义了两个宏：

int setjmp(jmp\_buf env); /\*设置调转点\*/

　　和

longjmp(jmp\_buf jmpb, int retval); /\*跳转\*/

　　宏setjmp的功能是将当前程序的状态保存在结构env ，为调用宏longjmp设置一个跳转点。setjmp将当前信息保存在env中供longjmp使用。其中env是jmp\_buf结构类型的，该结构定义为：

typedef struct {

unsigned j\_sp;

unsigned j\_ss;

unsigned j\_flag;

unsigned j\_cs;

unsigned j\_ip;

unsigned j\_bp;

unsigned j\_di;

unsigned j\_es;

unsigned j\_si;

unsigned j\_ds;

} jmp\_buf[1];

　　直接调用setjmp时，返回值为0，这一般用于初始化（设置跳转点时）。以后再调用longjmp宏时用env变量进行跳转。程序会自动跳转到setjmp宏的返回语句处，此时setjmp的返回值为非0，由longjmp的第二个参数指定。

　　下面通过例子来理解<setjmp.h>中定义的这两个宏。

　　例程9-1 非局部跳转演示。

#include <setjmp.h>

jmp\_buf env; /\*定义jmp\_buf类型变量\*/

int main(void)

{

int value;

value = setjmp(env); /\*调用setjmp，为longjmp设置跳转点\*/

if (value != 0)

{

printf("Longjmp with value %d\n", value);

exit(value); /\*退出程序\*/

}

printf("Jump ... \n");

longjmp(env,1); /\*跳转到setjmp语句处\*/

return 0;

}

　　本例程先应用setjmp宏为longjmp设置跳转点，当第一次调用setjmp时返回值为0，并将程序的当前状态（寄存器的相关状态）保存在结构变量env中。当程序执行到longjmp时，系统会根据setjmp保存下来的状态env跳转到setjmp语句处，并根据longjmp的第二个参数设置此时setjmp的返回值。

　　本例程的运行结果为：

Jump ...

Longjmp with value 1

　　一般地，宏setjmp和longjmp是成对使用的，这样程序流程可以从一个深层嵌套的函数中返回。

## 8. <signal.h>：信号

　　头文件<signal.h>中提供了一些处理程序运行期间引发的各种异常条件的功能，例如一些来自外部的中断信号等。

　　在<signal.h>中只定义了两个函数：

int signal(int sig, sigfun fname);

　　和

int raise(int sig);

　　signal函数的作用是设置某一信号的对应动作。其中参数sig用来指定哪一个信号被设置处理函数。在标准C 中支持的信号如下表。

标准C支持的信号

取值, 说明, 默认执行动作, 使用的操作系统

SIGABRT, 异常中止, 中止程序, UNIX DOS

SIGPPE, 算术运算错误, 中止程序, UNIX DOS

SIGILL, 非法硬件指令, 中止程序, UNIX DOS

SIGINT, 终端中断, 中止程序, UNIX DOS

SIGSEGV, 无效的内存访问, 中止程序, UNIX DOS

SIGTERM, 中止信号, 中止程序, UNIX DOS

参数fname是一个指向函数的指针，当sig的信号发生时程序会自动中断转而执行fname指向的函数。执行完毕再返回断点继续执行程序。系统提供了两个常量函数指针，可以作为函数的参数传递。它们分别是：

　　SIG\_DEF：执行默认的系统第一的函数。

　　SIG\_IGN：忽略此信号。

　　raise函数的作用是向正在执行的程序发送一个信号，从而使得当前进程产生一个中断而转向信号处理函数signal执行。其中参数sig为信号名称，它的取值范围同函数signal中的参数sig取值范围相同，见表9-6。

　　下面通过例子理解函数signal和raise。

　　例程9-2 signall和raise函数演示

#include <stdio.h>

#include <signal.h>

void Print1();

void Print2();

int main()

{ signal(SIGINT,Print1);

printf("Please enter Ctr+c for interupt\n") ;

getchar();

signal(SIGSEGV,Print2);

printf("Please enter any key for a interupt\n");

getchar();

raise(SIGSEGV);

}

void Print1()

{

printf("This is a SIGINT interupt!\n");

}

void Print2()

{

printf("This is a SIGSEGV interupt!\n");

}

　　本例程首先通过用户终端输入Ctrl+c产生一个终端中断，然后应用signal函数调用中断处理函数Print1；再通过raise函数生成一个无效内存访问中断，并通过signal函数调用中断处理函数Print2。

　　本例程的运行结果为：

Please enter Ctr+c for interupt

^C

This is a SIGINT interupt!

Please enter any key for a interupt

a

This is a SIGSEGV interupt!

## 9. <stdarg.h>：可变参数表

　　可变参数表<stdarg.h>中的宏是用来定义参数可变的函数的。在C语言中，有些库函数或者用户自定义的函数的参数是可变的，常用省略号“……”（例如库函数中的printf），定义这样的函数就要使用到<stdarg.h>中的宏。

　　1、va\_list

　　用于保存宏va\_start，va\_arg以及va\_end所需信息的数据类型。

　　2、<stdarg.h>中还定义了三个宏

void va\_start(va\_list ap, parmN);

type va\_arg(va\_list ap,type);

void va\_end (va\_list ap);

　　va\_start的作用是初始化ap，因此va\_start要在所有其它的va\_开头的宏前面最先使用(除了用va\_list定义变量外)，后面的va\_copy, va\_arg, va\_end都要使用到ap。在一对va\_start和va\_end之间不能再次使用va\_start宏。其中，parmN 为"..."之前的最后一个参数。例如, printf函数定义为：printf(const char \*format, ...); 那么在printf函数中的va\_start使用之后, parmN 的值就等于\*format。

　　va\_arg的作用就是返回参数列表ap中的下一个具有type类型的参数, 每次调用va\_arg都会修改ap的值, 这样才能连续不断地获取下一个type类型的参数。

　　va\_end与va\_start构成了一个scope，va\_end标志着结束，va\_end之后ap就无效了。

## 10. <stddef.h>：公共定义

在头文件<seddef.h>中，指定了标准库中的公共定义。其中主要包括以下内容：

1、NULL

空指针类型常量。

2、offset(type,member-designator)

　　它是扩展iz-t类型的一个整型常数表达式。它的值为从type定义的结构类型的开头到结构成员member-designator的偏移字节数。

3、ptrdiff\_t

　　表示两指针之差的带符号整数类型。

4、size\_t

　　表示由sizeof运算符计算出的结果类型，它是一个无符号整数类型。

5、wchar\_t

　　它是一种整数类型，取值范围为在被支持的地域环境中最大扩展字符集的所有字符的各种代码，空字符代码值为0。

## 11. <stdio.h>：输入输出

　　在头文件<stdio.h>中定义了输入输出函数，类型和宏。这些函数、类型和宏几乎占到标准库的三分之一。

　　下面给出头文件<stdio.h>中声明的函数以及功能简介，见下表。

<stdio.h>中声明的函数

　　函数定义, 函数功能简介

　　FILE \*fopen(char \*filename, char \*type), 打开一个文件

　　FILE \*fropen(char \*filename, char \*type,FILE \*fp), 打开一个文件，并将该文件关联到fp指定的流

　　int fflush(FILE \*stream), 清除一个流

　　int fclose(FILE \*stream), 关闭一个文件

　　int remove(char \*filename), 删除一个文件

　　int rename(char \*oldname, char \*newname), 重命名文件

　　FILE \*tmpfile(void), 以二进制方式打开暂存文件

　　char \*tmpnam(char \*sptr), 创建一个唯一的文件名

　　int setvbuf(FILE \*stream, char \*buf, int type, unsigned size), 把缓冲区与流相关

　　int printf(char \*format...), 产生格式化输出的函数

　　int fprintf(FILE \*stream, char \*format[, argument,...]), 传送格式化输出到一个流中

　　int scanf(char \*format[,argument,...]), 执行格式化输入

　　int fscanf(FILE \*stream, char \*format[,argument...]), 从一个流中执行格式化输入

　　int fgetc(FILE \*stream), 从流中读取字符

　　char \*fgets(char \*string, int n, FILE \*stream), 从流中读取一字符串

　　int fputc(int ch, FILE \*stream), 送一个字符到一个流中

　　int fputs(char \*string, FILE \*stream), 送一个字符到一个流中

　　int getc(FILE \*stream), 从流中取字符

　　int getchar(void), 从stdin流中读字符

　　char \*gets(char \*string), 从流中取一字符串

　　int putchar(int ch), 在stdout上输出字符

　　int puts(char \*string), 送一字符串到流中

　　int ungetc(char c, FILE \*stream), 把一个字符退回到输入流中

　　int fread(void \*ptr, int size, int nitems, FILE \*stream), 从一个流中读数据

　　int fwrite(void \*ptr, int size, int nitems, FILE \*stream), 写内容到流中

　　int fseek(FILE \*stream, long offset, int fromwhere), 重定位流上的文件指针

　　long ftell(FILE \*stream), 返回当前文件指针

　　int rewind(FILE \*stream), 将文件指针重新指向一个流的开头

　　int fgetpos(FILE \*stream), 取得当前文件的句柄

　　int fsetpos(FILE \*stream, const fpos\_t \*pos), 定位流上的文件指针

　　void clearerr(FILE \*stream), 复位错误标志

　　int feof(FILE \*stream), 检测流上的文件结束符

　　int ferror(FILE \*stream), 检测流上的错误

　　void perror(char \*string), 系统错误信息

　　在头文件<stdio.h>中还定义了一些类型和宏。

## 12. <stdlib.h>：实用函数

　　在头文件<stdlib.h>中声明了一些实现数值转换，内存分配等类似功能的函数。下面给出头文件<stdlib.h>中声明的函数以及功能简介，见下表。

<stdlib.h>中声明的函数

函数定义, 函数功能简介

double atof(const char \*s), 将字符串s转换为double类型

int atoi(const char \*s) , 将字符串s转换为int类型

long atol(const char \*s), 将字符串s转换为long类型

double strtod (const char\*s,char \*\*endp) , 将字符串s前缀转换为double型

long strtol(const char\*s,char \*\*endp,int base), 将字符串s前缀转换为long型

unsinged long strtol(const char\*s,char \*\*endp,int base), 将字符串s前缀转换为unsinged long型

int rand(void), 产生一个0~RAND\_MAX之间的伪随机数

void srand(unsigned int seed) , 初始化随机数发生器

void \*calloc(size\_t nelem, size\_t elsize), 分配主存储器

void \*malloc(unsigned size), 内存分配函数

void \*realloc(void \*ptr, unsigned newsize), 重新分配主存

void free(void \*ptr), 释放已分配的块

void abort(void), 异常终止一个进程

void exit(int status), 终止应用程序

int atexit(atexit\_t func), 注册终止函数

char \*getenv(char \*envvar), 从环境中取字符串

void \*bsearch(const void \*key, const void \*base, size\_t \*nelem, size\_t width, int(\*fcmp)(const void \*, const \*)), 二分法搜索函数

void qsort(void \*base, int nelem, int width, int (\*fcmp)()), 使用快速排序例程进行排序

int abs(int i), 求整数的绝对值

long labs(long n), 取长整型绝对值

div\_t div(int number, int denom), 将两个整数相除, 返回商和余数

ldiv\_t ldiv(long lnumer, long ldenom), 两个长整型数相除, 返回商和余数

有关上面列出的这些标准实用函数的功能、用法、例程等。

## 13. <time.h>：日期与时间函数

　　在头文件<time.h>中，声明了一些处理日期和时间的类型与函数。clock\_t和time\_t是两个表示时间值的算术类型。结构struct tm存储了一个日历时间的各个成分。结构tm的成员的意义及其正常的取值范围如下：

struct tm {

int tm\_sec; /\*从当前分钟开始经过的秒数(0,61)\*/

int tm\_min; /\*从当前小时开始经过的分钟数(0,59)\*/

int tm\_hour; /\*从午夜开始经过的小时数(0,23)\*/

int tm\_mday; /\*当月的天数(1,31)\*/

int tm\_mon; /\*从1月起经过的月数(0,11)\*/

int tm\_year; /\*从1900年起经过的年数\*/

int tm\_wday; /\*从本周星期天开始经过的天数(0,6)\*/

int tm\_yday; /\*从今年1月1日起经过的天数(0,356)\*/

int tm\_isdst; /\*夏令时标记\*/

};

　　如果夏令时有效，夏令时标记tm\_isdst值为正；若夏令时无效，tm\_isdst值为0；如果得不到夏令时信息，tm\_isdst值为负。

　　下面给出头文件<time.h>中声明的时间函数，见下表。

　　<time.h>中声明的时间函数

　　函数定义, 函数功能简介

　　clock\_t clock(void), 确定处理器时间函数

　　time\_t time(time\_t \*tp), 返回当前日历时间

　　double difftime(time\_t time2, time\_t time1), 计算两个时刻之间的时间差

　　time\_t mktime(struct tm \*tp), 将分段时间值转换为日历时间值

　　char \*asctime(const struct tm \*tblock), 转换日期和时间为ASCII码

　　char \*ctime(const time\_t \*time), 把日期和时间转换为字符串

　　struct tm \*gmtime(const time\_t \*timer), 把日期和时间转换为格林尼治标准时间(GMT)

　　struct tm \*localtime(const time\_t \*timer), 把日期和时间转变为结构

　　size\_t strftime(char \*s,size\_t smax,const char \*fmt, const struct tm \*tp) , 根据fmt的格式要求将\*tp中的日期与时间转换为指定格式。

# 第二章（IO函数）

[1. clearerr：复位错误标志函数 1](#_Toc278278674)

[2. feof：检测文件结束符函数 2](#_Toc278278675)

[3. ferror：检测流上的错误函数 3](#_Toc278278676)

[4. fflush：清除文件缓冲区函数 4](#_Toc278278677)

[5. fgetc：从流中读取字符函数 6](#_Toc278278678)

[6. fgetpos：取得当前文件的句柄函数 7](#_Toc278278679)

[7. fgets：从流中读取字符串函数 7](#_Toc278278680)

[8. fopen、fclose：文件的打开与关闭函数 8](#_Toc278278681)

[9. fprintf：格式化输出函数 10](#_Toc278278682)

[10. fputc：向流中输出字符函数 11](#_Toc278278683)

[11. fputs：向流中输出字符串函数 12](#_Toc278278684)

[12. fread：从流中读取字符串函数 12](#_Toc278278685)

[13. freopen：替换文件中数据流函数 13](#_Toc278278686)

[14. fscanf：格式化输入函数 14](#_Toc278278687)

[15. fseek：文件指针定位函数 15](#_Toc278278688)

[16. fsetpos：定位流上的文件指针函数 16](#_Toc278278689)

[17. ftell：返回当前文件指针位置函数 17](#_Toc278278690)

[18. fwrite：向文件写入数据函数 18](#_Toc278278691)

[19. getc：从流中读取字符函数 18](#_Toc278278692)

[20. getchar：从标准输入文件中读取字符函数 20](#_Toc278278693)

[21. gets：从标准输入文件中读取字符串函数 20](#_Toc278278694)

[22. perror：打印系统错误信息函数 21](#_Toc278278695)

[23. printf：产生格式化输出的函数 21](#_Toc278278696)

[24. putc：向指定流中输出字符函数 22](#_Toc278278697)

[25. putchar：向标准输出文件上输出字符 23](#_Toc278278698)

[26. puts：将字符串输出到终端函数 23](#_Toc278278699)

[27. remove：删除文件函数 24](#_Toc278278700)

[28. rename：重命名文件函数 25](#_Toc278278701)

[29. rewind：重置文件指针函数 25](#_Toc278278702)

[30. scanf：格式化输入函数 26](#_Toc278278703)

[31. setbuf、setvbuf：指定文件流的缓冲区函数 27](#_Toc278278704)

[32. sprintf：向字符串写入格式化数据函数 28](#_Toc278278705)

[33. sscanf：从缓冲区中读格式化字符串函数 29](#_Toc278278706)

[34. tmpfile：创建临时文件函数 29](#_Toc278278707)

[35. tmpnam：创建临时文件名函数 30](#_Toc278278708)

[36. ungetc：把字符退回到输入流函数 31](#_Toc278278709)

# clearerr：复位错误标志函数

函数原型：void clearerr(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：复位错误标志，即：使fp所指向的文件中的错误标志和文件结束标志置0。当输入输出函数对文件进行读写出错时，文件就会自动产生错误标志，这样会影响程序对文件的后续操作。clearerr函数就是要复位这些错误标志，也就是使fp所指向的文件的错误标志和文件结束标志置0，从而使文件恢复正常。

返回值：无

例程如下：复位错误标志演示。

#include <stdio.h>

int main(void)

{

FILE \*fp;

char ch;

/\* 以写的方式打开一个文件名为test.txt的文件 \*/

fp = fopen("test.txt", "w");

/\* 错误地从fp所指定的文件中读取一个字符，并打印它\*/

ch = fgetc(fp);

if (ferror(fp))

{

/\* 如果此操作错误，就发布错误信息\*/

printf("This is a error reading!\n");

/\*复位错误标志\*/

clearerr(fp);

}

/\*关闭文件\*/

fclose(fp);

return 0;

}

例程说明：

（1）首先程序以只写的方式打开一个文件名为”test.txt”的文件。这样，该文件就只能写而不能读了。

（2）程序企图应用fgetc函数从fp所指的文件中读出一个字符，这当然是违法的，因此文件自动产生错误标志。

（3）当用ferror函数检测出文件流存在错误时，就发布一条错误信息，并用clearerr函数清除fp指定的文件流所使用的错误标志，也就是使fp所指的文件的错误标志和文件结束标志置0。这样原先的错误就不会对文件的后续操作产生影响。

注意：ferror函数与clearerr函数应该配合使用。也就是说，通过ferror函数检测出文件有错误标志后要用clearerr函数复位错误标志。

# feof：检测文件结束符函数

函数原型：int feof(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：检测流上的文件结束符，即：检测文件是否结束。应用该函数可以判断一个文件是否到了结尾。在读取一个未知长度文件时，这个函数很有用。

返回值：遇到文件结束符返回非0，否则返回0。

例程如下：：检测文件结束标志演示。

#include <stdio.h>

int main(void)

{

FILE \*stream;

/\* 以只读方式打开test.txt文件 \*/

stream = fopen("test.txt", "r");

/\* 从文件中读取一个字符 \*/

fgetc(stream);

/\*检测是否是EOF，即结束标志 \*/

if (feof(stream))

printf("Have reached the end of the file!\n");

/\* 关闭该文件 \*/

fclose(stream);

return 0;

}

例程说明：

（1）首先程序打开一个名为test.txt的文件。

（2）应用fgetc函数从一个名为test.txt的文件中读取一个字符。

（3）判断它是否为文件结束标志EOF，如果是文件结束标志，就说明该文件已经结束，于是在屏幕上显示一条提示信息。如果不是文件的结束标志，就说明文件还未结束，信息不显示。

（4）最后关闭文件。

注意：在实际应用中，feof函数很重要，利用它程序员就可以很方便地判断当前的文件是否结束，从而进行不同的处理。例如，在从一个未知长度的文件中读取信息时，就可以利用feof函数判断什么时候该文件读完。

# ferror：检测流上的错误函数

函数原型：int ferror(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：检测流上的错误。即：检查文件在使用各种输入输出函数进行读写时是否出错。当输入输出函数对文件进行读写时出错，文件就会产生错误标志。应用这个函数，就可以检查出fp所指向的文件操作是否出错，也就是说是否有错误标志。

返回值：未出错返回值为0，否则返回非0，表示有错。

例程如下：应用ferror函数检查流上的错误。

#include <stdio.h>

int main(void)

{

FILE \*fp;

char ch;

/\* 以写的方式打开一个文件名为test.txt的文件 \*/

fp = fopen("test.txt", "w");

/\* 错误地从fp所指定的文件中读取一个字符，并打印它\*/

ch = fgetc(fp);

printf("%c\n",ch);

if (ferror(fp))

{

/\* 如果此操作错误，就发布错误信息\*/

printf("Error reading from test.txt !\n");

/\*复位错误标志\*/

clearerr(fp);

}

/\*关闭文件\*/

fclose(fp);

return 0;

}

例程说明：

（1）首先以只写的方式打开一个文件名为”test.txt”的文件。这样，该文件就只能写而不能读了。程序企图用fgetc函数从fp所指的文件中读出一个字符，这样就是非法操作，也就是说在用fgetc函数进行读取字符时出错了，因此文件产生错误标志。

（2）再用ferror函数来检测输入输出函数进行文件读写操作时是否出错，结果发现有错，因此函数返回一个非0整型数，并提示出错信息。

# fflush：清除文件缓冲区函数

函数原型：int fflush(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：清除一个流，即清除文件缓冲区，当文件以写方式打开时，将缓冲区内容写入文件。也就是说，对于ANSI C规定的是缓冲文件系统，函数fflush用于将缓冲区的内容输出到文件中去。

返回值：如果成功刷新，fflush返回0。指定的流没有缓冲区或者只读打开时也返回0值。返回EOF指出一个错误。

例程如下：第一种方式读写文件

#include <string.h>

#include <stdio.h>

#include <conio.h>

#include <io.h>

int main(void)

{

FILE \*stream1,\*stream2;

char test[20]="This is a test";

char res[20];

/\*以写的方式打开文件test.txt\*/

stream1 = fopen("test.txt", "w");

/\*向文件写入字符串\*/

fwrite(test,15,1,stream1);

/\*以读的方式打开文件test.txt\*/

stream2 = fopen("test.txt", "r");

/\*将文件内容读入缓冲区\*/

if(fread(res,15,1,stream2))

printf("%s",res);

else

printf("Read error!\n");

fclose(stream1);

fclose(stream2);

getch();

return 0;

}

例程如下：：第二种方式读写文件

#include <string.h>

#include <stdio.h>

#include <conio.h>

#include <io.h>

int main(void)

{

FILE \*stream1,\*stream2;

char test[20]="This is a test";

char res[20];

/\*以写的方式打开文件test.txt\*/

stream1 = fopen("test.txt", "w");

/\*向文件写入字符串\*/

fwrite(test,15,1,stream1);

/\*将缓冲区的内容写入文件\*/

fflush(stream1);

/\*以读的方式打开文件test.txt\*/

stream2 = fopen("test.txt", "r");

/\*将文件内容读入缓冲区\*/

if(fread(res,15,1,stream2))

printf("%s",res);

else

printf("Read error!\n");

fclose(stream1);

fclose(stream2);

getch();

return 0;

}

例程说明：

例程如下：中定义了两个文件指针stream1和stream2。

（1）首先以写的方式打开文件test.txt，用指针stream1指向该文件，并向文件中写入字符串"This is a test"。

（2）不关闭该文件，以读的方式打开文件test.txt，并用指针stream2指向该文件，试图将刚刚写入的字符串读入到内存缓冲区中。如果读入成功，打印出该字符串，否则报错。

实践证明，例程如下：的输出结果是在屏幕上显示错误信息Read error!。

例程如下：中定义了两个文件指针stream1和stream2。

（1）首先以写的方式打开文件test.txt，用指针stream1指向该文件，并向文件中写入字符串"This is a test"。

（2）调用fflush函数将缓冲区的内容写入文件。

（3）不关闭该文件，以读的方式打开文件test.txt，并用指针stream2指向该文件，试图将刚刚写入的字符串读入到内存缓冲区中。如果读入成功，打印出该字符串，否则报错。

实践证明，例程如下：的输出结果是在屏幕上显示字符串"This is a test"。

产生这样的效果原因在于：例程如下：中将文件打开后，指针stream1指向的是该文件的内存缓冲区，将字符串写入后也只是写到了文件的内存缓冲区中，而并没有写到磁盘上的文件中。而当以读的方式打开该文件时，该文件中的内容实际为空，也就是stream2指向的缓冲区中内容为空。因此读文件发生错误。而例程如下：中，在写完文件后调用函数fflush，将缓冲区的内容写到文件中，这样再以读的方式打开该文件时，文件中已经存有了字符串，因此可以正常读出。

注意：如果在写完文件后调用函数fclose关闭该文件，同样可以达到将缓冲区的内容写到文件中的目的，但是那样系统开销较大。

# fgetc：从流中读取字符函数

函数原型：int fgetc(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从流中读取字符，即从fp所指定的文件中取得下一个字符。这里需要注意，在每取完一个字符时fp会自动向下移动一个字节。这样编成时，程序员就不用再对fp控制了。这种功能在许多读写函数中都有体现。

返回值：返回所得到的字符，若读入错误。返回EOF。

例程如下：应用fgetc函数从文件中自动读取字符。

#include <string.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

FILE \*fp;

char string[] = "This is a test";

char ch;

/\* 以读写方式打开一个名为test.txt的文件 \*/

fp = fopen("test.txt", "w+");

/\* 向文件中写入字符串string \*/

fwrite(string, strlen(string), 1, fp);

/\* 将fp指针指向文件首 \*/

fseek(fp, 0, SEEK\_SET);

do

{

/\* 从文件中读一个字符 \*/

ch = fgetc(fp);

/\* 显示该字符 \*/

putch(ch);

} while (ch != EOF);

fclose(fp);

return 0;

}

例程说明：

（1）首先程序先以读写方式打开一个名为test.txt的文件，并向该文件中写入一个字符串。

（2）再应用fseek函数将文件指针fp定位在文件的开头，再循环地将字符逐一读出。这里每读出一个字符，指针fp会自动地向后移一个字节，直至读到文件尾，即EOF标志，循环才停止。因为fgetc函数的返回值为得到的字符，所以用一个字符型变量ch 来接受读出的字符。

（3）最后的运行结果是在屏幕上打印出This is a test字符串。

# fgetpos：取得当前文件的句柄函数

函数原型：int fgetpos( FILE \*stream, fpos\_t \*pos );

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：取得当前文件的指针所指的位置，并把该指针所指的位置数存放到pos所指的对象中。pos值以内部格式存储,仅由fgetpos和fsetpos使用。其中fsetpos的功能与fgetpos相反，为了详细介绍，将在后节给与说明。

返回值：成功返回0，失败返回非0，并设置errno。

例程如下：应用fgetpos函数取得当前文件的指针所指的位置。

#include <string.h>

#include <stdio.h>

int main(void)

{

FILE \*fp;

char string[] = "This is a test";

fpos\_t pos;

/\* 以读写方式打开一个名为test.txt的文件 \*/

fp = fopen("test.txt", "w+");

/\* 将字符串写入文件 \*/

fwrite(string, strlen(string), 1, fp);

/\* 取得指针位置并存入&pos所指向的对象 \*/

fgetpos(fp, &pos);

printf("The file pointer is at byte %ld\n", pos);

/\*重设文件指针的位置\*/

fseek(fp,3,0);

/\* 再次取得指针位置并存入&pos所指向的对象 \*/

fgetpos(fp, &pos);

printf("The file pointer is at byte %ld\n", pos);

fclose(fp);

return 0;

}

例程说明：

（1）首先，程序以读写方式打开一个名为test.txt的文件，并把字符串"This is a test"写入文件。注意：字符串共14个字节，地址为0~13。用fwrite函数写入后，文件指针自动指向文件最后一个字节的下一个位置。即这时的fp的值应该是14。

（2）再用fgetpos函数取得指针位置并存入&pos所指向的对象，此时， pos中的内容为14。然后在屏幕上显示出The file pointer is at byte 14。

（3）再用fseek函数重设文件指针的位置，让fp的值为3，即指向文件中第4个字节。

再次取得指针位置并存入&pos所指向的对象。然后在屏幕上显示出The file pointer is at byte 3。

# fgets：从流中读取字符串函数

函数原型：char \*fgets(char \*string, int n, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从fp所指的文件中读取一个长度为(n-1)的字符串，并将该字符串存入以string为起始地址的缓冲区中。fgets函数有三个参数，其中string为缓冲区首地址，n规定了要读取的最大长度，fp为文件指针。

返回值：返回地址string，若遇到文件结束符或出错，返回NULL。用feof 或ferror判断是否出错。

例程如下：用fgets函数从文件中读取字符串。

#include <string.h>

#include <stdio.h>

int main(void)

{

FILE \*fp;

char string[] = "This is a test";

char str[20];

/\* 以读写的方式打开一个名为test.txt的文件 \*/

fp = fopen("test.txt", "w+");

/\* 将字符串写入文件 \*/

fwrite(string, strlen(string), 1, fp);

/\* 文件指针定位在文件开头\*/

fseek(fp, 0, SEEK\_SET);

/\* 从文件中读一个长为strlen(string)的字符串 \*/

fgets(str, strlen(string)+1, fp);

/\* 显示该字符串 \*/

printf("%s", str);

fclose(fp);

return 0;

}

例程说明：

（1）首先，以读写的方式打开一个名为test.txt的文件，并将字符串写入文件。

应用fseek函数将文件指针定位在文件开头。

（2）从文件中读一个长为strlen(string)的字符串，这里应注意第二个参数若为n，则表示从fp所指的文件中读取一个长度为(n-1)的字符串。因此，这里的参数为strlen(string)+1，表示读取一个长度为strlen(string)的字符串。把字符串读到以str为首地址的数组中。

（3）最后显示该字符串。

# fopen、fclose：文件的打开与关闭函数

函数原型：FILE \*fopen(char \*filename, char \*type);

int fclose(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：函数fopen：打开一个流，即：打开一个文件。该函数有两个参数，filename是需要打开文件的文件名，type是打开文件的方式。函数fclose：关闭一个流，即：关闭一个文件，并释放文件缓冲区。fclose函数与fopen函数是相对的两个函数。fclose函数的参数是指向文件的指针，应用该函数用以在程序结束之前关闭文件，并释放文件缓冲区。这样可以保证文件的数据不流失。

在这里，特别列出所有的文件打开方式，以供大家参考。如下表所示。

文件的打开方式

|  |  |
| --- | --- |
| 文件使用方式 | 意 义 |
| r | 只读打开一个文本文件，只允许读数据 |
| w | 只写打开或建立一个文本文件，只允许写数据 |
| a | 追加打开一个文本文件，并在文件末尾写数据 |
| rb | 只读打开一个二进制文件，只允许读数据 |
| wb | 只写打开或建立一个二进制文件，只允许写数据 |
| ab | 追加打开一个二进制文件，并在文件末尾写数据 |
| r+ | 读写打开一个文本文件，允许读和写 |
| w+ | 读写打开或建立一个文本文件，允许读写 |
| a+ | 读写打开一个文本文件，允许读，或在文件末追加数据 |
| rb+ | 读写打开一个二进制文件，允许读和写 |
| wb+ | 读写打开或建立一个二进制文件，允许读和写 |
| ab+ | 读写打开一个二进制文件，允许读，或在文件末追加数据 |

返回值：fopen：FILE类型，如果打开的文件存在，返回指向该文件的指针；如果打开的文件不存在，则在指定的目录下建立该文件打开，并返回指向该文件的指针。fclose：整型，有错返回非0，否则返回0。

例程如下：打开并输出一个文件，然后关闭。

#include <string.h>

#include <stdio.h>

int main(void)

{

FILE \*fp;

char buf[11] = "abcdefghij";

/\* 以写方式打开文件名为test.txt的文件 \*/

fp = fopen("test.txt", "w");

/\*把字符串写入文件中\*/

fwrite(&buf, strlen(buf), 1, fp);

/\* 关闭文件 \*/

fclose(fp);

return 0;

}

例程说明：

（1）首先开辟一个11个字节大小的缓冲区buf，也就是数组，但预先只能存入10个字符。这是因为C语言中规定数组存放字符串时，最后一个字节要以’/0’结尾，作为结束标志，并由系统自动在字符串末尾添加’/0’标志。因此，11个字节大小的缓冲区只存放一个长10个字节的字符串。

（2）用fopen函数以写的方式打开一个名为test.txt的文件并将字符串写入文件。

调用fclose函数关闭该文件。

（3）fclose函数与fopen函数正好相对，其作用是关闭一个文件。当使用fopen函数打开一个文件时，会返回一个指向该文件的指针。在该例程中这个指针被赋值给fp，也就是说fp指向了test.txt这个文件。而当调用fclose函数关闭该文件，即fclose(fp)时，fp就不再指向该文件了，相应的文件缓冲区也被释放。

注意：用户在编写程序时应该养成及时关闭文件的习惯，如果不及时关闭文件，文件数据有可能会丢失。

# fprintf：格式化输出函数

函数原型：int fprintf(FILE \*fp, char \*format[, argument,...]);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：把argument的值以format所指定的格式输出到fp指向的文件中。这个函数理解起来和printf类似，在一般的使用中，第二个参数可以是一个字符串的头指针，也可以就是一个字符串。例如：fprintf(fp, "Cannot open this file!!")，意思就是把字符串Cannot open this file!!输出到文件fp中去。该函数一般用作终端的出错提示或是在磁盘中生成错误报告。

返回值：如果正确返回实际输出字符数，若错误则返回一个负数。

例程如下 用fprintf函数向终端发出出错提示。

#include <stdio.h>

int main(void)

{

FILE \*fp;

/\*以只读方式打开名为test.txt的文件\*/

if ((fp = fopen("\\test.txt", "rt"))

== NULL)

{

fprintf(stderr, "Cannot open this file!!\n");

return 1; /\*若该文件不能打开，在屏幕上显示出错提示\*/

}

/\*若该文件能够打开，在屏幕上显示正确提示\*/

fprintf(stderr,"Have open this file!!\n");

return 0;

}

例程说明：

（1）首先，以只读方式打开名为test.txt的文件，如果文件不能打开，这返回NULL。

（2）若该文件不能打开，在屏幕上显示出错提示。

（3）若该文件能够打开，在屏幕上显示正确提示。

注意：该函数中第一个参数是stderr，这是C语言中标准出错输出指针，它指向标准的出错输出文件，也就是显示器。因为，在操作系统中，I/O设备都是用文件进行管理的，因此设备都配有相应的控制文件。在C语言中，有三个文件与终端相联系，因此系统定义了三个文件指针。见下表：

|  |  |
| --- | --- |
| 设备文件 | 文件指针 |
| 标准输入 | stdin |
| 标准输出 | stdout |
| 标准出错输出 | stderr |

标准文件指针

在系统运行时，程序自动打开这三个标准文件。

本例程的运行结果为：

（1）如果不能打开文件：

Cannot open this file!!

（2）如果可以打开文件：

Have open this file!!

例程如下用fprintf函数在磁盘中生成错误报告。

#include <stdio.h>

int main(void)

{

FILE \*fp1,\*fp2;

/\*以只读方式打开名为test.txt的文件\*/

if ((fp1 = fopen("text.txt", "rt"))

== NULL)

{

/\*若文件打不开，则生成错误报告\*/

fp2=fopen("report.txt","w");

fprintf(fp2, "Cannot open this file!!\n");

return 1;

}

return 0;

}

例程说明：

（1）首先，以只读方式打开名为test.txt的文件，如果文件不能打开，这返回NULL。

（2）若该文件不能打开，则以写的方式打开一个名为report.txt的文件，并按照格式要求向文件中写入字符串"Cannot open this file!!\n"，即生成了一个错误报告。

注意：这里函数fprintf的第一个参数为文件指针，是用户自定义的，与上一例程的系统定义的文件指针stderr不同。fprintf函数与printf函数的使用类似，其实printf函数是fprintf函数的一个特例，printf函数只能向标准输出文件（显示器）输出数据，而fprintf函数也可以向一般用户定义的文件输出数据。

# fputc：向流中输出字符函数

函数原型：int fputc(char ch, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将字符ch输出到fp指向的文件中。该函数与前边提到的fgetc是相对的，第一个参数ch是字符型变量，函数将该变量中的字符输出到fp指向的文件中。

返回值：成功返回该字符，否则返回非0。

例程如下 应用fputc向文件输出字符。

#include <stdio.h>

int main(void)

{

/\*初始化字符数组str \*/

char str[] = "This is a test";

int i = 0;

/\*将数组中的字符循环输出至屏幕\*/

while (str[i])

{

fputc(str[i], stdout);

i++;

}

return 0;

}

例程说明：

（1）首先初始化字符数组str。这里应当知道，在C语言中初始化数组，系统会自动在数组最后添加"/0"，以表示该字符串结束。

（2）将数组中的字符循环输出至屏幕。这里注意两点：

* + - 该循环以"/0"作为结束标志，即循环碰到"/0"时结束。
    - 函数fputc的第二个参数是stdout，前面已讲过它代表标准输出文件指针，这样就是在屏幕上显示该字串。

本例程的运行结果为：

This is a test

# fputs：向流中输出字符串函数

函数原型：int fputs(char \*string, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将string所指的字符串输出到fp指向的文件中。该函数与fgets相对，第一个参数为字符串指针。与fgets函数不同的是，fputs函数没有字符串长度的限制，只是将string指向的字符串输出到文件中。

返回值：成功返回0，否则返回非0。

例程如下 应用fputs函数向文件中输出字符串。

#include <stdio.h>

int main(void)

{

FILE \*fp;

char str[]="This is a test!";

/\*以写的方式打开名为test.txt的文件\*/

fp=fopen("test.txt","w");

/\*将字符串写入文件\*/

fputs(str,fp);

fclose(fp);

return 0;

}

例程说明：

（1）首先用字符数组str存储一个字符串，并以写的方式打开名为test.txt的文件。

（2）再用fputs函数将该字符串输出到test.txt的文件中。

# fread：从流中读取字符串函数

函数原型：int fread(void \*buf, int size, int count, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从fp指向的文件中读取长度为size 的count个数据项，并将它输入到以buf为首地址的缓冲区中。此时，文件指针fp会自动增加实际读入数据的字节数，即fp指向最后读入字符的下一个字符位置。

返回值：返回实际读入数据项的个数，即count值。若遇到错误或文件结束返回0。

例程如下 应用fread函数从文件中读取数据到缓冲区。

#include <string.h>

#include <stdio.h>

int main(void)

{

FILE \*fp;

char str[] = "this is a test";

char buf[20];

if ((fp = fopen("test.txt", "w+"))

== NULL)

{

fprintf(stderr,

"Cannot open output file!!\n");

return 1;

}

/\* 向文件中写入字符串数组中的数据 \*/

fwrite(str, strlen(str), 1, fp);

/\* 将文件指针定位到文件开头 \*/

fseek(fp, SEEK\_SET, 0);

/\* 把文件中的数据读出并显示 \*/

fread(buf, strlen(str), 1, fp);

printf("%s\n", buf);

fclose(fp);

return 0;

}

例程说明：

（1）程序首先以读写方式打开名为test.txt的文件。这里有一个判断，若打开文件失败，则在屏幕上显示出错信息。

（2）应用fwrite函数向文件写入数据。有关fwrite函数后面做详细介绍。

（3）应用fseek函数将文件指针定位到文件开头。

（4）应用fread函数把文件中的数据读入内存。这里读取一个长度为strlen(str)的字符串，并将该字符串存入以buf为首地址的内存缓冲区中。

（5）显示该字符串。

# freopen：替换文件中数据流函数

函数原型：FILE \*freopen(char \*filename, char \*type, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：关闭fp所指向的文件，并将该文件中的数据流替换到filename所指向的文件中去。该函数共三个参数：第一个参数filename是文件流将要替换到的文件名路径；第二个参数type是文件打开的方式，它与fopen中的文件打开方式类似；第三个参数fp是要被替换的文件指针。

返回值：返回一个指向新文件的指针，即指向filename文件的指针。若出错，则返回NULL。

例程如下 关闭一个终端，并将数据流替换至一个新文件中。

#include <stdio.h>

int main(void)

{

FILE \*Nfp;

/\* 替换标准输出文件上的数据流到新文件test.txt \*/

if (Nfp=freopen("test.txt", "w", stdout)

== NULL)

fprintf(stderr, "error redirecting stdout\n");

/\* 标准输出文件上的数据流将会被替换到新文件中 \*/

printf("This will go into a file.");

/\* 关闭标准输出文件 \*/

fclose(stdout);

/\*关闭新生成的文件\*/

fclose(Nfp);

return 0;

}

例程说明：

（1）首先，程序以写的方式打开名为test.txt的文件，将标准输出文件上的数据流"This will go into a file."替换到新生成的文件test.txt中。freopen函数返回一个指向新文件的指针，即指向文件test.txt的指针，并将它存放到Nfp中。

（2）然后关闭标准输出文件，fclose(stdout)。

（3）最后关闭新生成的文件fclose(Nfp)。

（4）本程序的执行结果是在当前目录下生成一个文件test.txt，并将原终端的数据流"This will go into a file."重新写入test.txt文件中。

# fscanf：格式化输入函数

函数原型：int fscanf(FILE \*fp, char \*format[,argument...]);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从fp所指向的文件中按format给定的格式读取数据到argument所指向的内存单元。其中argument是指针，指针类型要与输入的格式format相一致。例如：fscanf(stdin, "%d", &i)；&i是整型的指针，输入的格式format也要为整型，即"%d"。

返回值：返回已输入的数据个数。若错误或文件结束返回EOF。

例程如下 应用fscanf函数从终端向内存输入数据。

#include <stdlib.h>

#include <stdio.h>

int main(void)

{

char ch;

printf("Please input an character ");

/\* 从标准输入文件中读取一个字符，并送至ch \*/

if (fscanf(stdin, "%c", &ch))

printf("The character was: %c\n",ch);

else

{

/\*出错提示\*/

fprintf(stderr, "Error reading an character from stdin!!\n");

exit(1);

}

return 0;

}

例程说明：

（1）首先，程序提示用户从键盘输入一个字符。这时，标准输入文件指针stdin指向该字符。

（2）调用fscanf函数从标准输入文件中读取一个字符，并送至ch。这里应该注意两点：

* + - fscanf函数的第一个参数不是用户定义的文件指针，而是系统定义的标准输入文件指针stdin。但用法与用户定义的文件指针类似。
    - &ch是指向字符型数据的指针，因此，输入的格式format也要为字符型"%c"，它们必须保持一致。

（3）该函数的使用与scanf类似。其实，scanf函数是fscanf函数的一个特例，它只能从标准输入文件（键盘终端）中输入数据。而fscanf函数则也可以从一般用户定义的文件中输入数据。

# fseek：文件指针定位函数

函数原型：int fseek(FILE \*fp, long offset, int base);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：重定位流上的文件指针，即将fp指向的文件的位置指针移向以base为基准，以offset为偏移量的位置。该函数有三个参数：fp为文件指针，offset为偏移量，即位移offset个字节，base为指针移动的基准，即起始点。其中，基准base用0、1或2表示。在ANSI C标准指定的名字如下表

ANSI C标准指定的起始点名字

|  |  |  |
| --- | --- | --- |
| 起始点 | 名字 | 数字代码 |
| 文件当前位置 | SEEK\_CUR | 1 |
| 文件开始位置 | SEEK\_SET | 0 |
| 文件末尾位置 | SEEK\_END | 2 |

偏移量用长整型数表示。ANSI C标准规定，在数的末尾加L就表示长整型数。该函数在随机读取较长的顺序文件时是很有用的。

返回值：成功返回0，否则返回非0。

例程如下 文件指针的定位演示。

#include <stdio.h>

void main( void )

{

FILE \*fp;

char line[81];

int result;

/\*以读写的方式打开打开名为test.txt的文件\*/

fp = fopen( "test.txt", "w+" );

if( fp == NULL )

printf( "The file test.txt was not opened！\n" );

else

{

/\*按照规定格式将字符串写入文件\*/

fprintf( fp, "The fseek begins here: "

"This is the file 'test.txt'.\n" );

/\*将文件指针定位到离文件头23个字节处\*/

result = fseek( fp, 23L, SEEK\_SET);

if( result )

perror( "Fseek failed" );

else

{

printf( "File pointer is set to middle of first line.\n" );

/\*从fp指向的文件中读取字符串\*/

fgets( line, 80, fp );

/\*显示读取的字符串\*/

printf( "%s", line );

}

fclose(fp);

}

}

例程说明：

（1）首先，程序以读写的方式打开打开名为test.txt的文件。

（2）然后，应用fprintf函数按照规定格式将字符串"The fseek begins here: ""This is the file 'test.txt'.\n"写入文件。

（3）再将文件指针定位到离文件头23个字节处，即将文件指针fp定位在字符串"This is the file 'test.txt'.\n"的开头。

（4）然后，应用fgets函数从fp指向的文件中读取字符串，并显示在屏幕上。

本程序的运行结果为在屏幕上显示出以下字符串：

File pointer is set to middle of first line.

This is the file 'test.txt'.

# fsetpos：定位流上的文件指针函数

函数原型：int fsetpos(FILE \*fp, const fpos\_t \*pos);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将文件指针定位在pos指定的位置上。该函数的功能与前面提到的fgetpos相反，是将文件指针fp按照pos指定的位置在文件中定位。pos值以内部格式存储,仅由fgetpos和fsetpos使用。

返回值：成功返回0，否则返回非0。

例程如下 应用fsetpos函数定位文件指针。

#include <stdio.h>

void main( void )

{

FILE \*fp;

fpos\_t pos;

char buffer[50];

/\*以只读方式打开名为test.txt的文件\*/

if( (fp = fopen( "test.txt", "rb" )) == NULL )

printf( "Trouble opening file\n" );

else

{

/\*设置pos值\*/

pos = 10;

/\*应用fsetpos函数将文件指针fp按照

pos指定的位置在文件中定位\*/

if( fsetpos( fp, &pos ) != 0 )

perror( "fsetpos error" );

else

{

/\*从新定位的文件指针开始读取16个字符到buffer缓冲区\*/

fread( buffer, sizeof( char ), 16, fp );

/\*显示结果\*/

printf( "16 bytes at byte %ld: %.16s\n", pos, buffer );

}

}

fclose( fp );

}

例程说明：

（1）首先，程序以只读方式打开名为test.txt的文件。在这里，test.txt文件中已存入字符串This is a test for testing the function of fsetpos.

（2）将pos设置为10。应用fsetpos函数将文件指针fp按照pos指定的位置在文件中定位。这样文件指针fp指向字符串中test的字母t。

（3）再从新定位的文件指针开始读取16个字符到buffer缓冲区，也就是说读取字符串"test for testing"到缓冲区buffer。

（4）最后显示结果：16 bytes at byte 10: test for testing 。

# ftell：返回当前文件指针位置函数

函数原型：long ftell(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：返回当前文件指针的位置。这个位置是指当前文件指针相对于文件开头的位移量。

返回值：返回文件指针的位置，若出错则返回–1L。

例程如下 应用ftell返回文件指针位置。

#include <stdio.h>

int main(void)

{

FILE \*fp;

fp = fopen("test.txt", "w+");

/\*按照格式要求将字符串写入文件\*/

fprintf(fp, "This is a test");

/\*读出文件指针fp的位置\*/

printf("The file pointer is at byte %ld\n", ftell(fp));

fclose(fp);

return 0;

}

例程说明：

（1）首先以写方式打开名为test.txt的文件，按照格式要求将字符串写入文件。注意：字符串共14个字符，地址为0~13。调用fprintf函数后，文件指针自动移到读入的最后一个字符的下一个位置，本例中就是文件的结束符，它的地址是14。

（2）应用ftell函数读出文件指针fp的位置。

注意：本题中ftell函数的返回值实际上就是该文件的长度。在实际的应用中，函数ftell常用来计算文件的长度。

# fwrite：向文件写入数据函数

函数原型：int fwrite(void \*buf, int size, int count, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将buf所指向的count\*size个字节的数据输出到文件指针fp所指向的文件中去。该函数与fread相对，输出数据后，文件指针fp自动指向输出的最后一个字符的下一个位置。该函数常用于分批将数据块输出到文件中。

返回值：返回实际写入文件数据项个数。

例程如下 应用fwrite函数向文件中写入数据块。

#include <stdio.h>

struct exp\_struct

{

int i;

char ch;

};

int main(void)

{

FILE \*fp;

struct exp\_struct s;

/\*以写的方式打开名为test.txt的文件\*/

if ((fp = fopen("test.txt","wb")) == NULL)

{

fprintf(stderr, "Cannot open the test.txt");

return 1;

}

/\*向结构体中的成员赋值\*/

s.i = 0;

s.ch = 'A';

/\* 将一个结构体数据块写入文件 \*/

fwrite(&s, sizeof(s), 1, fp);

fclose(fp);

return 0;

}

例程说明：

（1）程序先声明一个结构体类型struct exp\_struct，这样一个结构体变量就是一个小数据块。

（2）再以写的方式打开名为test.txt的文件。

（3）然后向结构体中的成员变量赋值，并将赋值好的数据块应用fwrite函数写入fp 所指向的文件中。这里参数sizeof(s)是该结构体变量的大小，1指只写入文件1个数据块。

（4）最后关闭该文件。

# getc：从流中读取字符函数

函数原型：int getc(FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从fp所指向的文件中读取一个字符。该函数与前面所讲到的fgetc作用类似。读取字符后，文件指针fp自动指向下一个字符。

返回值：返回所读的字符，若文件结束或出错返回EOF。

例程如下 应用getc函数从标准输入文件中读取一个字符。

#include <stdio.h>

int main(void)

{

char ch;

printf("Input a character:");

/\* 从标准输入文件中读取一个字符 \*/

ch = getc(stdin);

/\*显示该字符\*/

printf("The character input was: '%c'\n", ch);

return 0;

}

例程说明：

本程序是从标准输入文件（键盘）中读取一个字符，存入变量ch，并显示在屏幕上。

例程如下应用getc函数从一般文件中读取字符。

#include <stdio.h>

void main( void )

{

FILE \*fp;

char ch;

/\*以写的方式打开名为test.txt的文件\*/

fp=fopen("test.txt","w");

/\*写入字符串\*/

fprintf(fp,"This is a test.");

fclose(fp);

/\*再以读的方式打开名为test.txt的文件\*/

fp=fopen("test.txt","r");

/\*将文件指针指向文件开头\*/

fseek(fp,0L,SEEK\_SET);

/\*应用getc函数从文件中循环读取字符并显示出来\*/

while(feof(fp)==0)

{

ch=getc(fp);

printf("%c",ch);

}

fclose(fp);

return 0;

}

例程说明：

（1）首先以写的方式打开名为test.txt的文件，并将字符串"This is a test." 写入字符串。

（2）再以读的方式打开名为test.txt的文件，并将文件指针指向文件开头。

（3）最后，应用getc函数从文件中循环读取字符，直到文件结束为止，并将读取的字符显示到终端屏幕。

注意：本例程与上例不同，上例是从标准输入文件（键盘）中读取一个字符，本例是从一般文件中读取字符，关键在于函数的参数不同。

# getchar：从标准输入文件中读取字符函数

函数原型：int getchar(void);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从标准输入文件stdin（键盘）中读取一个字符。该函数与getc类似，但参数为空，它只能从标准输入文件中读取字符，而不能读取用户自定义的文件。getchar函数在编程时多用于接收回车、换行符。

返回值：返回所读的字符，若文件结束或出错返回-1。

例程如下 应用getchar函数从标准输入设备读取下一个字符。

#include <stdio.h>

int main(void)

{

int c;

/\* 从键盘上接收字符并显示，直到键入换行符为止 \*/

while ((c = getchar()) != '\n')

printf("%c", c);

return 0;

}

例程说明：

程序从键盘上接收字符并显示，当接收到换行符'\n'时，程序结束。

# gets：从标准输入文件中读取字符串函数

函数原型：char \*gets(char \*buf);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从标准输入文件stdin（键盘）中读取一个字符串，并把该字符串存入以buf为首地址的缓冲区中。该函数与fgets类似，但它只能从标准输入文件stdin中读取字符串，而且没有长度限制。

返回值：返回其参数，即缓冲区指针buf，若出错则返回空NULL。

例程如下 应用gets函数从键盘读取字符串。

#include <stdio.h>

int main(void)

{

char string[30];

printf("Input a string:");

/\*从终端输入字符串，注意不要超过30个字符\*/

gets(string);

/\*显示该字符串\*/

printf("The string input was: %s\n",string);

return 0;

}

例程说明：

（1）首先，程序开辟一个可容纳30个字符的字符串数组空间，并在屏幕上提示用户输入一个字符串。

（2）应用gets函数接收键盘输入的字符串，并把它存储到以string为首地址的缓冲区中。

（3）最后，将以string为首地址的缓冲区中的内容显示出来，即在屏幕上显示输入的字符串。

# perror：打印系统错误信息函数

函数原型：void perror(char \*string);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将错误信息输出到标准出错输出stderr。该函数的参数是字符串指针，指向错误信息字符串。也可以就是一个字符串，直接在参数中输入要显示的错误信息。但要注意，完整的错误信息不仅包括用户在参数中自己定义的字符串，还包括一个冒号，系统报错信息，和一个换行符。该函数主要用作向终端进行错误提示。

返回值：无。

例程如下 应用perror函数显示错误信息。

#include <stdio.h>

int main(void)

{

FILE \*fp;

/\*企图以读的方式打开文件test.txt\*/

fp = fopen("test.txt", "r");

if (fp==NULL)

/\*该文件不存在，在终端显示错误信息\*/

perror("Unable to open file for reading");

return 0;

}

例程说明：

（1）首先程序企图以读的方式打开文件test.txt，但在这里该文件并不存在。

（2） 然后，利用函数perror在终端显示错误信息"Unable to open file for reading: No such file or directory"。

具体的运行结果格式如下：

Unable to open file for reading: No such file or directory

注意：完整的错误信息包括：用户自定义字符串，冒号，系统报错信息，换行符。

# printf：产生格式化输出的函数

函数原型：int printf( const char \*format [, argument]... );

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：按format指向的格式字符串所规定的格式，将输出表列argument的值输出到标准输出设备。该函数与fprintf类似，但只能将argument的值输出到标准输出设备stdout,即显示器屏幕，而不能输出到用户自定义的文件中。

返回值：输出字符的个数，若出错则返回一个负数。

例程如下 应用printf函数输出字符串。

#include <stdio.h>

#include <string.h>

int main(void)

{

int a=1;

char ch='r';

char str[]="This is a test!";

printf("Output a string.\n");

printf("%s",str);

printf("The integer is %d\n",a);

printf("The character is %c\n",ch);

return 0;

}

例程说明：

（1）首先在标准输出设备stdout,即显示器屏幕上打印出"Output a string.\n"。

（2）再打印出字符串str中的内容："This is a test!"。

（3）再打印出整型数a：The integer is 1。

（4）再打印出字符ch：The character is r。

# putc：向指定流中输出字符函数

函数原型：int putc(int ch, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：向指定的文件输出一个字符。该函数有两个参数，ch是用户指定的字符，fp是文件指针。函数将字符ch输出到fp指定的文件中。

返回值：返回输出的字符ch，若出错则返回EOF。

例程如下 应用putc函数向标准输出文件输出字符。

#include <stdio.h>

int main(void)

{

char str[] = "Hello world!";

int i = 0;

while (str[i]){

putc(str[i], stdout);

i++;

}

return 0;

}

例程说明：

（1）首先将字符串"Hello world!"存入字符串数组str中。

（2）循环地将数组str中的内容输出到标准输出文件（显示器）上。

例程如下 应用putc函数向用户自定义文件输出字符。

#include <stdio.h>

int main(void)

{

FILE \*fp;

int i = 0;

char str[]="This is a test!";

fp=fopen("test.txt","w");

while (str[i]){

putc(str[i], fp);

i++;

}

fclose(fp);

return 0;

}

例程说明：

（1）首先将字符串"This is a test!" 存入字符串数组str中。

（2）以写的方式打开一个名为"test.txt"文件，并利用函数putc将数组str中的内容输出到文件"test.txt"中。

（3）关闭文件。

注意：该函数既可以向标准输出文件输出字符，又可以向用户自定义文件输出字符。而且，每当向文件输出一个字符时，文件指针就会自动向后移一个字节。

# putchar：向标准输出文件上输出字符

函数原型：int putchar(char ch);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将字符串ch输出到标准输出文件stdout上。也就是说将字符串ch输出到显示器屏幕上。

返回值：返回输出的字符ch，若出错，则返回EOF。

例程如下 应用putchar函数在屏幕上显示字符。

#include <stdio.h>

int main()

{

char str[]="This is a test!\n";

int i=0;

while(str[i]){

putchar(str[i]);

i++;

}

}

例程说明：

（1）首先将字符串"This is a test!\n"存入字符串数组str中。

（2）应用putchar函数，循环地将字符串输出到标准输出文件（终端屏幕）上。

注意：该函数与putc函数不同，它只能向标准输出文件，也就是终端屏幕上输出字符。而putc函数既可以向标准输出文件上输出字符，又可以向一般用户自定义文件上输出字符。

# puts：将字符串输出到终端函数

函数原型：int puts(char \*string);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将string指向的字符串输出到标准输出设备（stdout），并将'\0'转换为回车换行符'\n'。在C语言中，字符串以’\0’结尾。该函数不仅可以将字符串输出到标准输出设备，而且要将字符串的结束标志转换为回车换行。

返回值：成功则返回换行符，若失败则返回EOF。

例程如下 应用puts函数向终端输出字符串。

#include <stdio.h>

int main(void)

{

char string[] = "This is a test!\n";

puts(string);

return 0;

}

例程说明：

（1）首先，将字符串"This is a test!\n"存入以string为首地址的缓冲区。

（2）应用puts函数将该字符串显示在标准输出设备上。

注意：该函数将字符串的结尾标志'\0'转换为回车换行符'\n'。因此，程序运行的结果为：

This is a test!

# remove：删除文件函数

函数原型：int remove(char \*filename);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：删除以filename为文件名的文件。该函数的参数为欲删除文件的文件名，如果是单纯的文件名，就表明删除当前文件夹下的文件，否则要写明文件的路径。

返回值：成功删除文件返回0，否则返回-1。

例程如下 应用remove函数删除文件。

#include <stdio.h>

void main()

{

if( remove( "test.txt" ) == -1 )

perror( "Could not delete test.txt!!" );

else

printf( "Deleted test.txt \n" );

}

例程说明：

（1）首先要在当前文件夹下建立文件test.txt。

（2） 再利用remove函数删除该文件。若删除成功，则在终端显示字符串"Deleted test.txt \n"，否则显示字符串"Could not delete test.txt!!: No such file or directory"。

注意：前面已经讲过perror函数是按照一定格式要求向终端输出错误信息，因此，若删除文件成功，程序运行的结果为：

Deleted test.txt

若删除失败，则程序运行的结果为：

Could not delete test.txt!!: No such file or directory

这里，利用perror函数显示的完整的错误信息包括：用户自定义字符串，冒号，系统报错信息，换行符。

# rename：重命名文件函数

函数原型：int rename(char \*oldname, char \*newname);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：把由oldname所指的文件名改为由newname所指的文件名。该函数有两个参数，oldname为旧的文件名，newname为欲改成的新文件名。应当注意，oldname所指的文件一定要存在，newname所指的文件一定不存在。应用该函数可将一个文件的旧文件名oldname改为新文件名newname，但不能改变文件的路径。

返回值：成功返回0，出错返回-1。

例程如下 应用rename函数重命名文件。

#include <stdio.h>

int main(void)

{

if( rename("oldname.txt","newname.txt")==0)

printf("Rename successful!!");

else

printf("Rename fail!!");

}

例程说明：

（1）首先，在当前文件夹下建立一个文件"oldname.txt"。

（2）应用rename函数重命名该文件，将其改名为"newname.txt"。若重命名成功，在屏幕上显示"Rename successful!!"提示字符串；否则显示"Rename fail!!"提示字符串。

# rewind：重置文件指针函数

函数原型：void rewind(FILE \*stream);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将文件指针fp重新定位在文件开头，并清除文件的结束标志和错误标志。该函数与函数fseek功能类似，但不同的是，该函数可以清除文件的结束标志和错误标志，而函数fseek不能；另外，该函数不能像函数fseek一样返回一个值表明操作是否成功，因为该函数无返回值。

返回值：无。

例程如下 应用函数rewind重定位文件指针。

#include <stdio.h>

void main( void )

{

FILE \*fp;

int data1, data2;

data1 = 1;

data2 = 2;

if( (fp = fopen( "test.txt", "w+" )) != NULL )

{

fprintf( fp, "%d %d", data1, data2 );

printf( "The values written are: %d and %d\n", data1, data2 );

data1=0;

data2=0;

rewind( fp );

fscanf( fp, "%d %d", &data1, &data2 );

printf( "The values read are: %d and %d\n", data1, data2 );

fclose( fp );

}

}

例程说明：

（1）首先以写的方式打开一个名为"test.txt"的文件。

（2）应用fprintf函数向该文件写入data1，data2两个整型数，其值分别为1，2。此时，文件指针fp已指向文件尾。

（3）在屏幕上显示这两个数。

（4）再将变量data1和data2置0。

（4）应用rewind函数重定位文件指针，将文件指针fp重新定位在文件开头。

（5）再应用fscanf函数向data1，data2两个变量中读入文件中的数字。

（6）在屏幕上显示这两个数。

注意：在应用fprintf函数向该文件写入data1，data2两个整型数后，文件指针fp会自动指向文件尾。只有再应用函数rewind、fseek才能将文件指针重新定位到文件开头，以便读取文件。本例中，将data1和data2置0的目的是为了说明应用fscanf函数向data1，data2两个变量中读入文件中的数字的结果是正确的。本程序的运行结果为：

The values written are: 1 and 2

The values read are: 1 and 2

# scanf：格式化输入函数

函数原型：int scanf(char \*format[,argument,...]);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：从标准输入设备（键盘）按照format指定的格式字符串所规定的格式，将数据输入到argument所指定的内存单元。scanf函数与printf函数相对，前者是从标准输入设备输入数值，后者是从标准输出设备输出数值。

返回值：成功返回输入的字符个数，否则遇到结束符返回EOF，出错返回0。

例程如下 应用scanf函数输入数据。

#include <stdio.h>

void main( void )

{

int i;

char ch;

float f;

printf("Please input an integer:\n");

scanf("%d",&i);

getchar();

printf("Please input a character:\n");

scanf("%c",&ch);

getchar();

printf("Please input an float:\n");

scanf("%f",&f);

getchar();

printf("These values are:%d,%c,%f",i,ch,f);

}

例程说明：

（1）应用scanf函数向预先声明的三个变量空间输入一个整型数、一个字符、一个浮点数。

（2）在屏幕上显示这三个值。

注意：scanf函数与fscanf函数类似，但只能从标准输入设备文件读取数值，而不能像fscanf函数一样从一般用户自定义文件中读取数值。scanf函数常用作程序设计中数据的输入函数。

# setbuf、setvbuf：指定文件流的缓冲区函数

函数原型：void setbuf(FILE \*fp, char \*buf);

void setvbuf(FILE \*fp, char \*buf, int type, unsigned size);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能： 这两个函数使得打开文件后，用户可以建立自己的文件缓冲区，而不必使用fopen函数打开文件时设定的默认缓冲区。

setbuf函数的定义中，参数buf指定的缓冲区大小由stdio.h中定义的宏BUFSIZE的值决定，缺省值default为512字节。而当buf为NULL时，setbuf函数将使文件I/O不带缓冲区。而对setvbuf函数，则由malloc函数来分配缓冲区。参数size指明了缓冲区的长度(必须大于0)，而参数type则表明了缓冲的类型，其取值如下表：

setvbuf函数中参数type的取值含义

|  |  |
| --- | --- |
| type 值 | 含义 |
| \_IOFBF | 文件全部缓冲，即缓冲区装满后，才能对文件读写 |
| \_IOLBF | 文件行缓冲，即缓冲区接收到一个换行符时，才能对文件读写 |
| \_IONBF | 文件不缓冲，此时忽略buf,size的值，直接读写文件，不再经过文件缓冲区缓冲 |

返回值：无。

例程如下 应用setbuf函数指定文件的缓冲区。

#include <stdio.h>

void main( void )

{

char buf[BUFSIZ];

FILE \*fp1, \*fp2;

if( ((fp1 = fopen( "test1.txt", "a" )) != NULL) &&

((fp2 = fopen( "test2.txt", "w" )) != NULL) )

{

/\* 应用setbuf函数给文件流fp1指定缓冲区buf \*/

setbuf( fp1, buf );

/\*显示缓冲区地址\*/

printf( "fp1 set to user-defined buffer at: %Fp\n", buf );

/\* 文件流fp2不指定缓冲区\*/

setbuf( fp2, NULL );

/\*信息提示不分配缓冲区\*/

printf( "fp2 buffering disabled\n" );

fclose(fp1);

fclose(fp2);

}

}

例程说明：

（1）首先开辟一个大小为BUFSIZ的缓冲区，用作指定文件的缓冲区。这里，BUFSIZE为stdio.h中定义的宏，缺省值为512字节。

（2）以追加的方式和写的方式打开名为"test1.txt"和"test2.txt"的文件。

（3）应用setbuf函数给文件流fp1指定缓冲区buf，其中buf为缓冲区的首地址。并在屏幕上显示该首地址。

（4）文件流fp2不指定缓冲区，也就是第二个参数设置为NULL。并信息提示不分配缓冲区。

（5）关闭两个文件。

注意：使用setbuf函数指定文件的缓冲区时，一定要在文件读写之前。一旦用户自己指定了文件的缓冲区，文件的读写就要在用户指定的缓冲区中进行，而不在系统默认指定的缓冲区中进行。函数setvbuf的用法与setbuf类似，只是它的缓冲区大小可以动态分配，由函数的参数指定，而且缓冲区的类型也可以由参数指定。

# sprintf：向字符串写入格式化数据函数

函数原型：int sprintf(char \*string, char \*farmat [,argument,...]);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将格式化的数据存储到以string为首地址的缓冲区中。参数argument要被转化为farmat规定的格式，并按照这个规定的格式向字符串数组写入数据。这里应该注意，sprintf函数与printf函数和fprint函数不同，前者是向缓冲区（即数组）写入格式化数据，后者是向标准输出文件（stdout）和用户自定义文件输出格式化数据。

返回值：返回存储在string中数据的字节数。

例程如下 应用sprintf函数向指定缓冲区写入数据。

#include <stdio.h>

#include <math.h>

int main(void)

{

char str[80];

sprintf(str, "An approximation of Pi is %f\n", M\_PI);

puts(str);

return 0;

}

例程说明：

（1）首先开辟一个80字节大小的缓冲区str，即：该缓冲区以str为首地址。

（2）再将指定的字符串写入缓冲区str中。其中，M\_PI是math.h中定义的常量3.141593。

（3）应用puts函数向终端输出该字符串。

注意：puts函数的作用是把str指定的字符串输出到标准输出设备，并且将字符串结束标志'/0'转换为回车换行符。因此，该程序运行的结果是：

An approximation of Pi is 3.141593

除了规定字符串中的换行符外，程序还将'/0'转换为换行符。

# sscanf：从缓冲区中读格式化字符串函数

函数原型：int sscanf(char \*string, char \*format[,argument,...]);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：将string指定的数据读到argument所指定的位置。其中，参数argument是与format格式要求相符合的变量指针。也就是说，如果format指定的格式为"%d"，则argument就必须是整型变量的指针。这里应该注意，sscanf函数与scanf函数和fscanf函数不同，前者是从指定的缓冲区读格式化数据到新的缓冲区中，而后者是从标准输入文件（stdin）和用户自定义文件中读取格式化数据到缓冲区中。

返回值：成功返回已分配空间的数量，返回0表示没用空间分配，返回EOF表示出错。

例程如下 应用sscanf函数读取格式化数据。

#include <stdio.h>

void main( void )

{

char str[] = "1 2 3...";

char s[81];

char c;

int i;

float fp;

/\* 从缓冲区str中读取数据 \*/

sscanf( str, "%s", s );

sscanf( str, "%c", &c );

sscanf( str, "%d", &i );

sscanf( str, "%f", &fp );

/\* 输出已读取的数据 \*/

printf( "String = %s\n", s );

printf( "Character = %c\n", c );

printf( "Integer: = %d\n", i );

printf( "Real: = %f\n", fp );

}

例程说明：

（1）首先开辟一个以str为首地址的缓冲区，并初始化其内容。

（2）应用sscanf函数从缓冲区str中读取数据。分别以格式要求"%s"、"%c"、"%d"、"%f"读取，并存入相应的变量中。

（3）输出已读取的数据。

# tmpfile：创建临时文件函数

函数原型：FILE \*tmpfile(void);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：创建一个临时文件。该文件以w+b（二进制读写）方式打开，当该文件被关闭时，该文件会被自动删除。

返回值：返回指向临时文件的指针，如果文件打不开则返回EOF。

例程如下 创建一个临时文件。

#include <stdio.h>

#include <process.h>

int main(void)

{

FILE \*tempfp;

tempfp = tmpfile();

if (tempfp)

printf("Temporary file be created!!\n");

else

{

printf("Unable to create the temporary file!!\n");

exit(1);

}

sleep(20);

return 0;

}

例程说明：

（1）首先应用tmpfile函数创建一个临时文件，并将文件指针赋值给FILE型变量tempfp。

（2）如果创建临时文件成功，则在终端显示提示：Temporary file be created!!

（3）如果创建不成功，则显示提示：Unable to create the temporary file!!

（4）程序挂起20秒。

注意：这里将程序挂起20秒的目的是为了让用户看到生成的临时文件。这是因为当程序执行完时，系统会自动删除临时文件，那样用户就感觉不到临时文件的创建了。当该程序运行时，会在当前文件夹下生成一个临时文件。

临时文件的作用是暂时存储程序运行过程中需要的数据，当程序运行完毕时，这些数据也就没有用了。

# tmpnam：创建临时文件名函数

函数原型：char \*tmpnam(char \*string);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：创建一个临时文件名，用以打开一个临时文件。

返回值：创建成功返回指向该文件名的指针，否则返回NULL。

例程如下

#include <string.h>

#include <stdio.h>

main()

{

char tmp[10];

tmpnam(tmp);

printf("The temporary name is %s\n",name);

}

例程说明：

（1）首先定义一个字符型数组tmp。

（2）调用tmpnam函数生成一个临时文件名。

（3）打印出该临时文件名。

注意：应用函数tmpnam生成的临时文件名是不同于任何已存在文件名的有效文件名。本函数用于给临时文件创建文件名。

# ungetc：把字符退回到输入流函数

函数原型：int ungetc(char c, FILE \*fp);

头文件：#include<stdio.h>

是否是标准函数：是

函数功能：把字符c退回到fp所指向的文件流中，并清除文件的结束标志。fp所指向的文件既可以是用户自定义的文件，又可以是系统定义的标准文件。

返回值：成功返回字符c，否则返回EOF。

例程如下 应用ungetc函数向标准输入文件退回字符。

#include <stdio.h>

#include <ctype.h>

int main( void )

{

int i=0;

char ch;

puts("Input an integer followed by a char:");

/\* 读取字符直到输入非数字或EOF \*/

while((ch = getchar()) != EOF && isdigit(ch))

i = 10 \* i + ch - 48; /\* 将ASCII码转换为整数值\*/

/\* 如果输入非数字，将其退回输入流 \*/

if (ch != EOF)

ungetc(ch, stdin);

printf("i = %d, next char in buffer = %c\n", i, getchar());

return 0;

}

例程说明：

（1）首先从终端输入一串字符串，要求输入整型数字，并以非数字字符结尾。

（2）程序读取字符，直到输入非数字或EOF为止，并将数字字符串转换为整型数。例如：将字符串"123"转换为整型数123。

（3）再将结尾的非数字字符退回到标准输入文件（stdin）。

（4）显示退回到标准输入文件中的字符。

注意：所谓将结尾的非数字字符退回到标准输入文件，就是说将数字字符串后面的那个非数字字符退回到标准输入文件中去。例如：输入的字符串为"123abc"，那么退回的字符就是a。这样，程序将前面的字符串"123"转换为整型数123，并存入变量i中。a作为数字字符串输入的结束标志（因为 a是继数字字符之后的第一个非数字字符），被退回到标准输入文件（stdin）。于是，再调用getchar函数读取的字符就应该是刚刚退回到标准输入文件中的字符a。因此，本段例程的执行结果为：

Input an integer followed by a char:

123abc

i = 123, next char in buffer = a

即：输入的字符串中123 为整型数，接下来的字符为a。

## 第三章（字符处理函数）

[1. isalnum：检查字符是否是字母或数字 1](#_Toc278280384)

[2. isalpha：检查字符是否是字母 2](#_Toc278280385)

[3. isascii：检查字符是否是ASCII码 3](#_Toc278280386)

[4. iscntrl：检查字符是否是控制字符 3](#_Toc278280387)

[5. isdigit：检查字符是否是数字字符 4](#_Toc278280388)

[6. isgraph：检查字符是否是可打印字符（不含空格） 5](#_Toc278280389)

[7. islower：检查字符是否是小写字母 5](#_Toc278280390)

[8. isprint：检查字符是否是可打印字符（含空格） 6](#_Toc278280391)

[9. ispunct：检查字符是否是标点字符 6](#_Toc278280392)

[10. isspace：检查字符是否是空格符 7](#_Toc278280393)

[11. isupper：检查字符是否是大写字母 8](#_Toc278280394)

[12. isxdigit：检查字符是否是十六进制数字字符 9](#_Toc278280395)

[13. toascii：将字符转换为ASCII码 9](#_Toc278280396)

[14. tolower：将大写字母转换为小写字母 10](#_Toc278280397)

[15. toupper：将小写字母转换为大写字母 11](#_Toc278280398)

# isalnum：检查字符是否是字母或数字

函数原型：int isalnum( int c );

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c 是否是字母（alpha）或数字（number）。

返回值：是字母或数字返回1，否则返回0。

例程如下： 应用isalnum检查字符属性。

#include <stdio.h>

#include <ctype.h>

int main( void )

{

char c,ch;

scanf("%c",&c);

ch=getchar();

while(c!='e') {

if(isalnum(c))

printf("This is a alpha or a number\n");

else

printf("This is a particulate character\n");

scanf("%c",&c);

ch=getchar();

}

return 1;

}

例程说明：

（1）首先，程序声明了两个字符型变量，用以接收来自终端的字符。

（2）当用户输入的字符不是'e'且是字母或数字字符时，就在屏幕上显示"This is a alpha or a number"提示信息。当用户输入的字符不是'e'且不是字母或数字字符时，就在屏幕上显示"This is a particulate character"提示信息。

（3）当用户输入字符'e'时，程序退出。

注意：本例程中，scanf函数用以接收欲判断的字符，getchar函数用以接收回车换行符。本例程的运行结果为：

a

This is a alpha or a number

2

This is a alpha or a number

#

This is a particulate character

e

# isalpha：检查字符是否是字母

函数原型：int isalpha( int c );

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c 是否是字母（alpha）。

返回值：是字母返回1，否则返回0。

例程如下：应用isalpha检查字符属性。

#include <stdio.h>

#include <ctype.h>

int main( void )

{

char c,ch;

scanf("%c",&c);

ch=getchar();

while(c!='e') {

if(isalpha (c))

printf("This is a alpha \n");

else

printf("This is not a alpha\n");

scanf("%c",&c);

ch=getchar();

}

return 1;

}

例程说明：

本例程但只判断输入的字符是否是字母，如果是字母，则在屏幕上显示"This is a alpha "提示信息，否则显示"This is not a alpha"提示信息。

本例程的运行结果为：

a

This is a alpha

3

This is not a alpha

$

This is not a alpha

e

# isascii：检查字符是否是ASCII码

函数原型：int isascii(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c 是否是ASCII码，所谓ASCII码是指0x00~0x7F之间的字符。

返回值：是ASCII码返回1，否则返回0。

例程如下：应用isascii检查字符属性。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

int c;

c='A';

printf("%c:%s\n",c,isascii(c)?"yes":"no");

c=0x7f;

printf("%c:%s\n",c,isascii(c)?"yes":"no");

c=0x80;

printf("%c:%s\n",c,isascii(c)?"yes":"no");

getchar();

return 0;

}

例程说明：

本例程应用isascii函数判断字符'A'、0x7f、0x80是否是ASCII码，如果是，显示"yes",不是则显示"no"。本例程的运行结果是：

A:yes

⌂:yes

Ç:no

注意：所谓ASCII码是指0x00~0x7F之间的字符，本例程中十六进制数0x7f的字符显示为⌂，属于ASCII码，因此显示yes；0x80的字符显示为Ç，不属于ASCII码，因此显示no。

# iscntrl：检查字符是否是控制字符

函数原型：int iscntrl( int c );

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c 是否是控制字符，控制字符的ASCII码在0到0x1F之间。

返回值：是控制字符返回1，否则返回0。

例程如下：应用iscntrl检查字符属性。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char c,ch;

printf("Input some character until contrl character\n");

scanf("%c",&c);

ch=getchar();

while(!iscntrl(c)){

scanf("%c",&c);

ch=getchar();

};

return 0;

}

例程说明：

输入的字符不是控制字符时，可以一直输入下去，一旦输入了控制字符，程序结束。

注意：每输入一个字符时，要以回车结束。

本例程的运行结果为：

a

b

# isdigit：检查字符是否是数字字符

函数原型：int isdigit( int c );

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c 是否是数字字符（0~9）。

返回值：是数字字符返回1，否则返回0。

例程如下：应用isdigit函数统计字符串中数字个数。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char ch;

int i=0;

ch=getchar();

while(ch!=EOF){

if(isdigit(ch))i++;

ch=getchar();

}

printf("%d",i);

}

例程说明：

（1）首先，程序中设置字符型变量ch用以接收输入的字符，设置整型变量i，并初始化i=0，用以统计输入的字符串中数字的个数。

（2）当输入的字符不是EOF时，程序循环执行，并应用isdigit函数判断用户输入的字符是否是数字字符，如果是则在变量i上加1。

（3）最后显示输入的字符串中数字个数。

注意：利用Ctrl+Z组合键输入的字符就是EOF。

本例程的运行结果为：

abc123def567ghi^Z

6

# isgraph：检查字符是否是可打印字符（不含空格）

函数原型：int isgraph(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c是否是除了空格符外的可打印字符，其ASCII码在0x21-0x7e之间。

返回值：是除了空格符外的可打印字符返回1，否则返回0。

例程如下： 应用isgraph函数判断可打印字符。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

int c;

c='A';

printf("%c:%s\n",c,isgraph(c)?"yes":"no");

c=' ';

printf("%c:%s\n",c,isgraph(c)?"yes":"no");

c=0x7f;

printf("%c:%s\n",c,isgraph(c)?"yes":"no");

getchar();

return 0;

}

例程说明：

本例程应用isgraph函数判断字符'A'、' '、0x7f是否是除了空格符外的可打印字符。如果是，显示"yes",不是则显示"no"。本例程的运行结果是：

A:yes

:no

⌂:no

# islower：检查字符是否是小写字母

函数原型：int islower(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c是否是小写字母（a~z）。

返回值：当c为小写字母时，返回1，否则返回0。

例程如下： 应用islower函数统计字符串中的小写字母个数。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char ch;

int i=0;

ch=getchar();

while(ch!=EOF){

if(islower(ch))i++;

ch=getchar();

}

printf("%d",i);

getchar();

return 0;

}

例程说明：

本例程先输入一串任意的字符，然后应用islower函数统计字符串中的小写字母个数。最后，在屏幕上显示出小写字母的个数。本例程的运行结果是：

djcvGGJH4623^Z

4

注意：^Z是Ctrl+Z组合键的屏幕显示，即结束标志EOF。

# isprint：检查字符是否是可打印字符（含空格）

函数原型：int isprint(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c是否为可打印字符（含空格），其ASCII码在0x20-0x7e之间。

返回值：是可打印字符返回1，否则返回0。

例程如下：应用isprint函数判断可打印字符。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

int c;

c='A';

printf("%c:%s\n",c,isprint(c)?"yes":"no");

c=' ';

printf("%c:%s\n",c,isprint(c)?"yes":"no");

c=0x7f;

printf("%c:%s\n",c,isprint(c)?"yes":"no");

getchar();

return 0;

}

例程说明：

本例程与例程11-6相似，应用isprint函数判断字符'A'、' '、0x7f是否是可打印字符（包括空格）。如果是，显示"yes",不是则显示"no"。本例程的运行结果是：

A:yes

:yes

⌂:no

# ispunct：检查字符是否是标点字符

函数原型：int ispunct(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c是否是除字母、数字、空格之外的可打印字符，也就是检查字符c是否是标点字符。

返回值：当c为标点符号时，返回1，否则返回0。

例程如下： 应用ispunct函数判断标点字符。

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int main(void)

{

char s[]="He said:Oh!Very well!";

int i;

printf("%s\n",s);

for(i=0;i<strlen(s);i++)

{

if(ispunct(s[i])) printf("^");

else printf(".");

}

return 0;

}

例程说明：

（1）首先，将字符串"He said:Oh!Very well!"存入以s为首地址的缓冲区中，并在屏幕上显示该字符串。

（2）循环检查该字符串中的每个字符，并在屏幕上显示的该字符串下方作出标记，即：如果不是标点字符，打印"."，如果是标点字符，打印"^"。本例程的运行结果是：

He said:Oh!Very well!

. . . . . . .^.. ^. . . . . . . .^

# isspace：检查字符是否是空格符

函数原型：int isspace(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c是否为空格符space、制表符tab或是换行符。空格符space的ASCII码为32，制表符tab的ASCII码为9，换行符的ASCII码则为

返回值：当c为空格符或制表符时，返回1，否则返回0。

例程如下： 应用isspace函数转换空格符、制表符和换行符。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char s[]="space |NewLine\n|table\t|";

int i;

printf("%s",s);

printf("\n");

for(i=0;i<strlen(s);i++)

{

if(isspace(s[i])) putchar('.');

else putchar(s[i]);

}

getchar();

return 0;

}

例程说明：

（1）首先，将字符串"space |NewLine\n|table\t|"存入以s为首地址的缓冲区中，并在屏幕上显示该字符串。其中，' '、'\n'、'\t'分别为空格符、换行符、制表符，在屏幕上显示其字符原样。

（2）再通过isspace函数检测出该字符串中的这些空格符、换行符、制表符，将其转换为'.'字符，并输出到终端屏幕。

注意：本例程并没有改变原字符串数组中的存储内容，只是在输出时将字符串中的空格符、换行符、制表符转换为'.'字符并输出到终端屏幕。本例程的运行结果是：

space |NewLine

|table |

space.|NewLine.|table.|

# isupper：检查字符是否是大写字母

函数原型：int isupper(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c是否是大写字母（A~Z）。

返回值：当c为大写字母时，返回1，否则返回0。

例程如下： 应用isupper函数统计字符串中的小写字母个数。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char ch;

int i=0;

ch=getchar();

while(ch!=EOF){

if(isupper(ch))i++;

ch=getchar();

}

printf("%d",i);

getchar();

return 0;

}

例程说明：

本例程利用函数isupper统计输入的字符串中大写字母的个数。最后，在屏幕上显示出小写字母的个数。本例程的运行结果是：

ABCDEabcFG123^Z

7

# isxdigit：检查字符是否是十六进制数字字符

函数原型： int isxdigit(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：检查字符c是否为十六进制数字。

返回值：当c为A-F,a-f或0-9之间的十六进制数字时，返回非零值，否则返回0。

例程如下：应用isxdigit函数检查字符属性。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char c;

c='f';

printf("%c:%s\n",c,isxdigit(c)?"yes":"no");

c='1';

printf("%c:%s\n",c,isxdigit(c)?"yes":"no");

c='$';

printf("%c:%s\n",c,isxdigit(c)?"yes":"no");

getchar();

return 0;

}

例程说明：

本例程应用isxdigit函数判断字符'f'、'1'、'$'是否是十六进制数字，如果是，显示"yes",不是则显示"no"。本例程的运行结果是：

f:yes

1:yes

$:no

# toascii：将字符转换为ASCII码

函数原型：int toascii(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：将c转化为相应的ASCII码。

返回值：返回转换后的数值，也就是转换后的ASCII码。

例程如下： 应用toascii函数将整型数字转换为相应的ASCII码。

#include <stdio.h>

#include <ctype.h>

main()

{

int s[]={1,2,3,4,5,6};

int i;

for(i=0;i<6;i++)

{

printf("%d",s[i]);

}

printf("\n");

for(i=0;i<6;i++)

{

putchar(toascii(s[i]));

}

getchar();

return 0;

}

例程说明：

（1）首先，在整型数组中存入1~6六个整型数字，并将其显示在终端屏幕上。

（2）循环地将数组中的每个数字转换为其对应的ASCII码，并将其以字符的形式显示在终端屏幕上。本例程的运行结果为：

1 2 3 4 5 6

☺☻♥ ♦ ♣ ♠

# tolower：将大写字母转换为小写字母

函数原型：int tolower(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：将c转化为相应的小写字母。

返回值：如果c为大写英文字母，则返回对应的小写字母；否则返回原来的值。

例程如下： 应用tolower函数将大写字母转换为小写字母。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char str[]="This Is A Test!";

int i;

printf("%s\n",str);

for(i=0;i<strlen(str);i++)

{

putchar(tolower(str[i]));

}

getchar();

return 0;

}

例程说明：

（1）首先，将字符串"This Is A Test!"存入以str为首地址的缓冲区中，并将该字符串显示在终端屏幕上。

（2）应用"This Is A Test!"函数将该字符串中大写字母转换为小写字母，并输出。本

注意：本例程将字符串中大写字母转换为小写字母并输出，但并不改变原数组中的内容，只是在输出时将大写字母转换为小写字母，而本身是小写字母的字符或非字母字符，则返回原值。本例程的运行结果是：

This Is A Test!

this is a test!

# toupper：将小写字母转换为大写字母

函数原型：int toupper(int c);

头文件：#include<ctype.h>

是否是标准函数：是

函数功能：将c转化为相应的大写字母。

返回值：如果c为小写英文字母，则返回对应的大写字母；否则返回原来的值。

例程如下： 应用toupper函数将小写字母转换为大写字母。

#include <stdio.h>

#include <ctype.h>

int main(void)

{

char str[]="This Is A Test!";

int i;

printf("%s\n",str);

for(i=0;i<strlen(str);i++)

{

putchar(toupper(str[i]));

}

getchar();

return 0;

}

例程说明：

本例程利用toupper函数将字符串中的小写字母转换为大写字母，并输出到终端。本例程的运行结果为：

This Is A Test!

THIS IS A TEST!

## 第四章（字符串函数）

[1. atof：字符串转浮点型函数 1](#_Toc278280694)

[2. atoi：字符串转整型函数 2](#_Toc278280695)

[3. atol：字符串转长整型函数 3](#_Toc278280696)

[4. memchr：字符搜索函数 3](#_Toc278280697)

[5. memcmp：字符串比较函数 4](#_Toc278280698)

[6. memcpy： 字符串拷贝函数 5](#_Toc278280699)

[7. memmove： 字块移动函数 6](#_Toc278280700)

[8. memset：字符加载函数 7](#_Toc278280701)

[9. strcat：字符串连接函数 8](#_Toc278280702)

[10. strchr：字符串中字符首次匹配函数 9](#_Toc278280703)

[11. strcmp：字符串比较函数 10](#_Toc278280704)

[12. strcpy：字符串拷贝函数 11](#_Toc278280705)

[13. strcspn：字符集逆匹配函数 11](#_Toc278280706)

[14. strdup：字符串新建拷贝函数 12](#_Toc278280707)

[15. strerror：字符串错误信息函数 13](#_Toc278280708)

[16. strlen：计算字符串长度函数 14](#_Toc278280709)

[17. strlwr：字符串小写转换函数 15](#_Toc278280710)

[18. strncat：字符串连接函数 16](#_Toc278280711)

[19. strncmp：字符串子串比较函数 16](#_Toc278280712)

[20. strncpy：字符串子串拷贝函数 18](#_Toc278280713)

[21. strpbrk：字符集字符匹配函数 19](#_Toc278280714)

[22. strrchr：字符串中字符末次匹配函数 20](#_Toc278280715)

[23. strrev：字符串倒转函数 21](#_Toc278280716)

[24. strset：字符串设定函数 21](#_Toc278280717)

[25. strspn：字符集匹配函数 22](#_Toc278280718)

[26. strstr：字符串匹配函数 23](#_Toc278280719)

[27. strtod：字符串转换成双精度函数 24](#_Toc278280720)

[28. strtok：字符串分隔函数 25](#_Toc278280721)

[29. strtol：字符串转换成长整型函数 26](#_Toc278280722)

[30. strtoul：字符串转换成无符号长整型函数 27](#_Toc278280723)

[31. strupr：字符串大写转换函数 28](#_Toc278280724)

[32. strupr：字符串大写转换函数 29](#_Toc278280725)

# atof：字符串转浮点型函数

函数原型：float atof(const char \*str);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：将字符串转换成浮点值，也就是将字符串str转换成浮点值然后获取转换后的结果。

返回值：返回转换后的浮点值

例程如下： 应用atol将字符串转换成浮点值。

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

char \*str="12345.67";

float result;

result=atof(str);

printf("string=%s\nfloat =%f\n",str,result);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串作为待转换的字符串，声明的浮点型变量result用于获取转换结果。

（2）程序通过调用atol将字符串转换为相应的浮点型变量，获取转换结果，转换规则与strtoX函数相同。

（3）最后将转换结果打印出来。

本例程的运行结果是：

string =12345.67

float=12345.669922

注意：本例程中，转换成浮点数的结果有些奇怪，它并不等于我们字符串中变量的值，而是存在一定的误差，虽然误差很小，但是可以看出误差是从原字符串中的最后一位开始的，这是由于在转换过程中函数内部在实现时采用的转换方式造成的，如果想避免这种误差，可以使用strtoX系列函数。

# atoi：字符串转整型函数

函数原型：int atoi(const char \*str);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：将字符串转换成整数值，也就是将字符串str转换成整型值然后获取转换后的结果。

返回值：返回转换后的整型值

例程如下： 应用atoi将字符串转换成整型值。

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

char \*str="12345.67";

int result;

result=atoi(str);

printf("string=%s\ninteger=%d\n",str,result);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串作为待转换的字符串，声明的整型变量result用于获取转换结果。

（2）程序通过调用atoi将字符串转换为相应的整型变量，获取转换结果，转换规则与strtoX函数相同。

（3）最后将转换结果打印出来。

本例程的运行结果是：

string =12345.67

integer=12345

# atol：字符串转长整型函数

函数原型：long atol(const char \*str);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：将字符串转换成长整数值，也就是将字符串str转换成长整型值然后获取转换后的结果。

返回值：返回转换后的长整型值

例程如下： 应用atol将字符串转换成长整型值。

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

char \*str="12345.67";

long result;

result=atol(str);

printf("string=%s\nlong =%ld\n",str,result);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串作为待转换的字符串，声明的长整型变量result用于获取转换结果。

（2）程序通过调用atol将字符串转换为相应的长整型变量，获取转换结果，转换规则与strtoX函数相同。

（3）最后将转换结果打印出来。

本例程的运行结果是：

string =12345.67

long=12345

# memchr：字符搜索函数

函数原型：void \*memchr(void \*s, char ch, unsigned n)

头文件：#include<string.h>

是否是标准函数：是

函数功能：在数组的前n个字节中搜索字符 ch。

返回值：返回一个指针，它指向ch在s 中第一次出现的位置。如果在s的前n个字符中找不到匹配，返回NULL。

例程12.31应用函数memchr搜索一个字符串的子串。

#include <string.h>

#include <stdio.h>

int main(void)

{

char \*str="I love China\n";

char \*p;

p=memchr(str,'C',strlen(str));

if(p)

printf("%s",p);

else

printf("The character was not found\n") ;

}

例程说明：

（1）首先初始化字符串“I love China\n”，将首地址赋值给str。

（2）在字符串str中查找字符’C’出现的位置，并返回以第一个字符’C’开头的字符子串的指针。

（3）如果返回值不为NULL，打印该子串。

本例程的运行结果为：

China

# memcmp：字符串比较函数

函数原型：void \*memcmp(char \*s1, char \*s2, unsigned n)

头文件：#include<string.h>

是否是标准函数：是

函数功能：比较s1所指向的字符串与s2所指向的字符串的前n个字符。

返回值：根据s1所指向的对象的大于、等于、小于s2所指向的对象，函数memcmp分别返回大于、等于、小于0的值。

例程如下：比较两个字符串。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*str1="ABCDEF";

char \*str2="ABCDEf";

int s1,s2;

s1=memcmp(str1,str2,6);

s2=memcmp(str1,str2,5);

printf("The comparison of 6 character\n");

if(s1>0)printf("%s>%s\n",str1,str2);

else

if(s1<0)printf("%s<%s\n",str1,str2);

else

printf("%s=%s\n",str1,str2);

printf("The comparison of 5 character\n");

if(s2>0)printf("%s>%s\n",str1,str2);

else

if(s2<0)printf("%s<%s\n",str1,str2);

else

printf("%s=%s\n",str1,str2);

}

例程说明：

（1）首先初始化两个字符串“ABCDEF”和“ABCDEf”。

（2）然后应用函数memcmp将这两个字符串按照不同的字符个数进行比较，将返回的比较结果复制给变量s1和s2。

（3）显示比较结果。

本例程的运行结果为：

The comparison of 6 character

ABCDEF<ABCDEf

The comparison of 5 character

ABCDEF=ABCDEf

注意：

由于字符串比较的方法是从左至右按照字符的ASCII码进行比较的，因此在比较6个字符时，字符串“ABCDEF”<“ABCDEf”（f的ASCII值大于F的ASCII值）；而只比较5个字符时，字符串“ABCDEF”=“ABCDEf”。

# memcpy： 字符串拷贝函数

函数原型：void \*memcpy(void \*destin, void \*source, unsigned n)

头文件：#include<string.h>

是否是标准函数：是

函数功能：从source所指的对象中复制n个字符到destin所指的对象中。但是，如果这种复制发生在重叠对象之间，其行为是不可预知的。

返回值：destin

例程如下：利用函数memcpy进行字符串拷贝。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*s = "##########";

char \*d = "This is a test for memcpy function";

char \*ptr;

printf("destination before memcpy: %s\n", d);

ptr = memcpy(d, s, strlen(s));

if (ptr)

printf("destination after memcpy: %s\n", d);

else

printf("memcpy failed\n");

return 0;

}

例程说明：

（1）首先定义两个字符串s和d，并赋初值，且d的长度大于s。

（2）显示字符串d的原始内容。

（3）通过函数memcpy将字符串s复制到字符串d中，并返回字符串d的首指针。

（4）如果拷贝成功，再次显示字符串d的内容。

本例程的运行结果为：

destination before memcpy: This is a test for memcpy function

destination after memcpy: ##########test for memcpy function

注意：

1、memcpy与strcpy的不同在于应用memcpy进行字符串的拷贝可以指定拷贝串的长度。另外memcpy的参数为void指针类型，因此它还可以对非字符型对象进行操作，而strcpy只适用于字符串的拷贝。

2、前面提到，如果复制过程中发生在重叠对象之间，其行为是不可预知的。例如下面这个例子：

#include <string.h>

#include <stdio.h>

int main(void)

{

char \*d = "1234567890";

char \*p;

p=d+3;

printf(" %s\n", d);

memcpy(p, d, 6);

printf(" %s\n", d);

return 0;

}

由于字符串p是字符串d的一个子串，在调用memcpy时，复制的字符串在d和p之间又重叠，因此该复制行为是不可预知的，结果也自然难以保证。这段程序的运行结果为：

1234567890

1231231230

显然这不是期望得到的结果。

# memmove： 字块移动函数

函数原型：void \*memmove(void \*destin, void \*source, unsigned n)

头文件：#include<string.h>

是否是标准函数：是

函数功能：从source所指的对象中复制n个字符到destin所指的对象中。与memcpy不同的是，当对象重叠时，该函数仍能正确执行。

返回值：destin

例程如下：利用函数memmove进行字符块的移动

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*s = "##########";

char \*d = "This is a test for memcpy function";

char \*ptr;

printf("destination before memmove: %s\n", d);

ptr = memmove(d, s, strlen(s));

if (ptr)

printf("destination after memmove: %s\n", d);

else

printf("memcpy failed\n");

return 0;

}

例程说明：

（1）首先定义两个字符串s和d，并赋初值，且d的长度大于s。

（2）显示字符串d的原始内容。

（3）通过函数memmove将字符串s复制到字符串d中，并返回字符串d的首指针。

（4）如果拷贝成功，再次显示字符串d的内容。

本例程的运行结果为：

destination before memmove: This is a test for memcpy function

destination after memmove: ##########test for memcpy function

注意：

与函数memcpy不同的是，当对象重叠时，该函数仍能正确执行。例如下面这个例子：

#include <string.h>

#include <stdio.h>

int main(void)

{

char \*d = "1234567890";

char \*p;

p=d+3;

printf(" %s\n", d);

memmove(p, d, 6);

printf(" %s\n", d);

return 0;

}

虽然复制的字符串在d和p之间又重叠，但本段程序的运行结果为：

1234567890

1231234560

显然这是期望得到的结果。

这是因为函数memmove的复制行为类似于先从source对象中复制n个字符到一个与source和destin都不重合的含n个字符的临时数组中作为缓冲，然后从临时数组中再复制n个字符destin所指的对象中。

就本段程序而言，memmove先将字符串“123456”复制到一个临时数组中，再将它复制到以p为首地址的字符串中。

# memset：字符加载函数

函数原型：void \*memset(void \*s, int c, unsigned n)

头文件：#include<string.h>

是否是标准函数：是

函数功能：把c复制到s所指向的对象的前n个字符的每一个字符中。

返回值：s的值

例程如下：应用memset函数替换字符串中的字符。

#include <string.h>

#include <stdio.h>

int main(void)

{

char \*str="AAAAAAAAAAAAAAAAAA";

printf("The original string is: %s\n",str);

memset(str,'B',9);

printf("The string after memset is:%s\n",str);

}

例程说明：

（1）首先初始化字符串“AAAAAAAAAAAAAAAAAA”， 将首地址赋值给str。

（2）显示该字符串。

（3）利用函数memset将字符串str的前9个字符替换为’B’。

（4）显示替换后的字符串。

本例程的运行结果为：

The original string is: AAAAAAAAAAAAAAAAAA

The string after memset is:BBBBBBBBBAAAAAAAAA

# strcat：字符串连接函数

函数原型：char \*strcat (char \*dest,char \*src);

头文件：#include<string.h>

是否是标准函数：是

函数功能：将两个字符串连接合并成一个字符串，也就是把字符串src连接到字符串dest后面，连接后的结果放在字符串dest中

返回值：指向字符串dest的指针

例程如下： 应用strcat连接字符串。

#include <string.h>

#include <stdio.h>

int main( )

{

  char dest[20]={“ ”};

char \*hello = "hello ", \*space = " ", \*world = "world";

  strcat(dest, hello);

   strcat(dest, space);

strcat(dest, world);

   printf("%s\n", destination);

getch();

   return 0;

}

例程说明：

（1）首先，程序声明了一个字符数组和三个字符串变量，将字符数组dest初始化位空串，其余三个字符串变量分别赋予初值。

（2）程序通过调用strcat函数实现字符串的连接，首先将字符串hello添加到字符数组dest的末端，此时字符数组dest的值有空串变为"hello"，然后继续调用两次strcat函数，依次将字符串space和字符串world陆续连接到字符数组dest的末端，从而完成整个字符串的连接操作。

（3）最后将最终的结果输出。

本例程的运行结果是：

hello world

注意：本例程中，开始对字符数组dest初始化位空是必要的，对声明的变量进行初始化是一个很好的习惯，如果不对字符数组dest进行初始化程序会产生运行时的错误，有兴趣的读者可以试试未初始化程序的输出结果。

# strchr：字符串中字符首次匹配函数

函数原型：char \*strchr(char \*str, char c);

头文件：#include<string.h>

是否是标准函数：是

函数功能：在字符串中查找给定字符的第一次匹配，也就是在字符串str中查找字符c第一次出现的位置

返回值：第一次匹配位置的指针

例程如下： 应用strchr匹配字符串中字符。

#include <stdio.h>

#include <string.h>

int main(void)

{

char str[15] ={""};

char \*ptr, c = 'r';

strcpy(str, "Hello World");

ptr = strchr(str, c);

if (ptr)

printf("The character %c is at position: %d\n", c, ptr-str);

else

printf("The character was not found\n");

strcpy(str, "Aloha");

if (ptr)

printf("The character %c is at position: %d\n", c, ptr-str);

else

printf("The character was not found\n");

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串和一个字符数组以及一个字符，并对字符变量赋予了我们要查找的值。

（2）程序通过调用strcpy 赋予了字符数组一个值，然后调用strchr函数在字符数组中查找第一次与字符变量c匹配的字符，也就是查找第一个'r' 字符，返回的结果为指向第一个匹配字符的指针。根据返回值输出匹配结果。

（3）程序第二次通过调用strcpy 赋予了字符数组一个值，然后调用strchr函数在字符数组中查找第一次与字符变量c匹配的字符，也就是查找第一个'r' 字符，最后根据返回值输出匹配结果。

（4）第二次匹配已经给字符串重新赋值，我们理解新的字符串似乎应该是"Aloha"，从而没有与'r'匹配的字符，但实际的运行结果却令人大吃一惊。这时因为在重新赋值时"Aloha"虽然将"Hello"覆盖掉，但是后面的字符仍然在数组中保留，因此在做匹配的时候仍然得到与第一次匹配相同的结果。

（5）对结果进行输出时，如果匹配成功，那么我们输出匹配的字符在数组中的位置，如果匹配不成功，则显示没有找到。

本例程的运行结果是：

The character r is at position 8

The character r is at position 8

注意：本例程中，对字符串中字符匹配的返回值是指向匹配位置的指针，我们获取到该指针后，与数组的头指针做减法，也就是与数组变量名做减法，就可以获得我们得到的指针在数组中对应的下标。

# strcmp：字符串比较函数

函数原型：int strcmp (char \*str1,char \* str2);

头文件：#include<string.h>

是否是标准函数：是

函数功能：比较两个字符串的大小，也就是把字符串str1和字符串str2从首字符开始逐字符的进行比较，直到某个字符不相同或比较到最后一个字符为止，字符的比较为ASIC码的比较

返回值：若字符串str1大于字符串str2返回结果大于零，若字符串str1小于字符串str2返回结果小于零，若字符串str1等于字符串str2返回结果等于零

例程如下： 应用strcmp比较字符串大小。

#include <string.h>

#include <stdio.h>

int main(void)

{

char \*str1 = "Canada", \*str2 = "China", \*str3 = "china";

int result;

result = strcmp(str1, str2);

if (result < 0)

printf("%s is less than %s", str1,str2);

else

printf("%s is not less than %s", str1,str2);

printf("\n");

result = strcmp(str2, str3);

if (result < 0)

printf("%s is less than %s", str2,str3);

else

printf("%s is not less than %s", str2,str3);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了三个字符串变量并分别赋予了初值，注意字符串str2和字符串str3的区别在于首字母是否大写，整型变量result用于记录字符串的比较结果。

（2）程序通过调用strcmp函数比较字符串str1和字符串str2，在首字符相同的情况下第二个字符' a'的ASIC码小于' h'的ASIC码，因此比较结果为字符串str1小于字符串str2，返回结果小于零。第二次调用strcmp函数比较字符串str2和字符串str3，由于在ASIC码表中小写字母在后，小写字母的ASIC码大于大写字母，即' C'小于' c'，因此比较结果为字符串str2小于字符串str3，返回结果小于零。

（3）最后将最终的结果输出，为了使输出结果一目了然，在两次比较中间的printf函数输出了一个换行。

本例程的运行结果是：

Canada is less than China

China is less than china

注意：本例程中，字符串的比较结果为首个两个不等字符之间ASIC码的差值，如果我们将第一次比较的结果result输出，应该是' a'的ASIC与码与' h'的ASIC码的差值，有兴趣的读者可以试试输出结果。

# strcpy：字符串拷贝函数

函数原型： char \* strcpy (char \*dest,char \* src);

头文件：#include<string.h>

是否是标准函数：是

函数功能：实现字符串的拷贝工作，也就是把字符串src中的内容拷贝到字符串dest中，使两个字符串的内容相同。

返回值：指向字符串dest的指针

例程如下： 应用strcpy实现字符串拷贝。

#include <stdio.h>

#include <string.h>

int main(void)

{

char dest[20] ={""};

char \*src = "Hello World";

int result;

strcpy(dest,src);

printf("%s\n", dest);

result=strcmp(dest,src);

if(!result)

printf("dest is equal to src");

else

printf("dest is not equal to src");

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串和一个字符数组并给分别赋予了初值，整型变量result用于记录字符串子串的比较结果。

（2）程序通过调用strcpy函数将字符串src中的内容拷贝到字符数组dest中，使得两者具有相同的内容。为了验证两个变量中的内容是否真的一样，通过调用strcmp对两个字符串中的内容进行比较。

（3）最后将拷贝结果和比较结果输出。

本例程的运行结果是：

Hello World

dest is equal to src

注意：本例程中，向字符数组中赋值时要保证字符数组中有足够的空间，虽然有时候即便空间不够也会打印出正确的结果，但随着程序的运行，不能保证超出下标范围的部分还能以正确的型式存在。

# strcspn：字符集逆匹配函数

函数原型：int strcspn(char \*str1, char \*str2);

头文件：#include<string.h>

是否是标准函数：是

函数功能：在字符串中查找第一个属于字符集的下标，即从开始有多少个字符不属于字符集，也就是在字符串str1中查找第一个属于字符集str2中任何一个字符的下标，即字符串str1中从开始一直有多少个字符不属于字符集str2中的字符。

返回值：所找到的字符串中段的长度

例程如下： 应用strspn逆匹配字符串中字符集。

#include <string.h>

#include <stdio.h>

int main(void)

{

char \*str1="tomato",\*str2="carrot";

char \*str= "abc";

int result;

result = strcspn(str1,str);

if(result)

printf("The first %d is congruent\n",result);

else

printf("No character is congruent\n");

result = strcspn(str2,str);

if(result)

printf("The first %d is congruent\n",result);

else

printf("No character is congruent\n");

getch();

return 0;

}

例程说明：

（1）首先，程序声明了三个字符串并分别赋予初值，其中最后一个变量是用于逆匹配的字符集。

（2）程序通过调用strcspn进行字符集的逆匹配，它从字符串str1中的第一个字符开始检查是不是属于字符串str中的任意字符，如果不属于就继续逆匹配，直到逆匹配不成功，本例中会发现直到遇到字符'a'才逆匹配失败，因为字符'a'属于字符串str中的某个字符。然后输出匹配结果。

（3）程序第二次通过调用strspn对字符串str2进行字符集逆匹配，发现第一个字符即属于字符集str中的某字符。

（4）输出匹配结果是显示前多少个字符逆匹配成功。

本例程的运行结果是：

The first 3 is congruent

No character is congruent

注意：本例程中，字符集逆匹配与字符集匹配两者的匹配方式截然相反，字符串匹配是当字符串中字符等于字符集中任意字符是匹配成功，字符串逆匹配是当字符串中字符不等于字符集中任意字符是匹配成功。

# strdup：字符串新建拷贝函数

函数原型：char \*strdup(char \*str);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：将字符串拷贝到新分配的空间位置，也就是将str拷贝到一块新分配的存储空间，其内部使用动态分配内存技术实现的，分配给字符串的空间来自于当前所用内存模式制定的堆。

返回值：返回指向含有该串拷贝的存储区

例程如下： 应用strdup将字符串拷贝到新建位置处。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*src="This is the buffer text";

char \*dest;

dest=strdup(src);

if(!strcmp(src,dest))

printf("Copy success\n%s\n",dest);

else

printf("Copy failure");

free(dest);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了两个字符串并给第一个字符串赋于初值，此时并未给字符串dest分配任何空间。

（2）程序通过调用strdup将字符串拷贝到新建位置处，通过动态分配内存技术将新分配一个与字符串src大小相同的存储区并完成字符串的复制工作，然后返回该存储区并让dest指向该区域。

（3）程序通过调用strcmp比较复制前后的字符串，如果复制成功而这应当相同，函数返回值为零，并打印拷贝结果。

（4）由于新分配的存储区是通过动态分配内存技术实现的，因此在程序退出之前要将分配的存储区显示的释放。

本例程的运行结果是：

Copy success

This is the buffer text

注意：本例程中，初学者往往会忽视释放动态分配存储区的操作，虽然表面看起来似乎对程序没有什么影响，但实际上不对存储区进行回收会造成内存泄漏，在一些大程序会造成致命的后果。

# strerror：字符串错误信息函数

函数原型：char \*strerror(int errnum);

头文件：#include<string.h>

是否是标准函数：是

函数功能：获取程序出现错误的字符串信息，也就是根据错误代码errnum查找到具体的错误信息。

返回值：返回错误信息

例程如下： 应用strerror查看几种错误信息。

#include <stdio.h>

#include <errno.h>

int main(void)

{

char \*error;

int i;

for(i=0;i<12;i++)

{

error=strerror(i);

printf("%s",error);

}

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串用于获取错误信息，声明的整型变量既作为循环变量又作为错误信息代码。

（2）程序通过调用strerror根据错误代码获取到具体的错误信息，其中这些代表具体错误信息的字符串在相应的头文件中定义。循环只取了前十二种错误信息，实际的错误种类还有更多。

（3）每次循环将具体错误信息打印出来。

本例程的运行结果是：

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

注意：本例程中，如果读者有兴趣，不妨看看一共系统定义了多少种错误信息，通过更改循环变量将各种错误信息打印出来。

# strlen：计算字符串长度函数

函数原型： int strlrn (char \*str);

头文件：#include<string.h>

是否是标准函数：是

函数功能：求字符串的长度，也就是求字符串str中有多少个字符

返回值：字符串str字符的个数

例程如下： 应用strlen求字符串长度。

#include <stdio.h>

#include <string.h>

int main(void)

{

char src1[3]={""},src2[10]={"Hello"};

char \*src3="Hello";

printf("%d\n",strlen(src1));

printf("%d\n",strlen(src2));

printf("%d\n",strlen(src3));

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串和两个字符数组并给分别赋予了初值，我们将字符串src3与字符数组src2赋予相同的初值。

（2）程序通过调用strlen函数分别求出三个变量字符的长度。在求字符长度时，返回的结果是有效字符的个数，因此虽然字符数组src1由十个字符变量组成，但初值为空串，因此长度为零，并不等于数组长度。由于字符串src3与字符数组src2赋予相同的初值，因此两者长度相同。

（3）最后将字符串或字符数组的长度值输出。

本例程的运行结果是：

0

5

5

注意：本例程中，如果将字符数组src2拷贝到字符数组src1中，并不会产生任何编译错误，但是程序运行时会产生不可预知的结果，有兴趣的读者可以试试完成拷贝后将三个变量的长度输出。

# strlwr：字符串小写转换函数

函数原型：char \*strlwr(char \*str,);

头文件：#include<string.h>

是否是标准函数：否

函数功能：将字符串原有大写字符全部转换为小写字符，也就是将字符串str中的所有字符变成小写。

返回值：返回指向被转换字符串的指针

例程如下： 应用strlwr将字符串转换成小写字符。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*s="You'll Never Walk Alone";

printf("%s",strlwr(s));

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串为待转换字符串并赋予初值。

（2）程序通过调用strlwr将字符串中的所有大写字符转换成小写字符，并返回转换后的结果。

（3）最后将转换结果打印出来。

本例程的运行结果是：

you'll never walk alone

# strncat：字符串连接函数

函数原型：char \*strncat (char \*dest, char \*src, int n);

头文件：#include<string.h>

是否是标准函数：是

函数功能：将一个字符串的子串连接到另一个字符串末端，也就是把字符串src的前n个字符连接到字符串dest后面，连接后的结果放在字符串dest中

返回值：指向字符串dest的指针

例程如下：应用strncat连接字符串子串。

#include <string.h>

#include <string.h>

#include <stdio.h>

int main(void)

{

char dest[30]={""};

char \*favorite = "I love", \*tabs = "\t\n", \*language = "C++";

strcnat(dest, favorite,6);

strncat(dest, tabs,1);

strncat(dest, language,1);

printf("%s\n", dest);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符数组和三个字符串变量，将字符数组dest初始化位空串，其余三个字符串变量分别赋予初值，其中字符串tans由两个字符组成，一个制表符和一个换行符。

（2）程序通过调用strncat函数实现字符串子串的连接，首先将字符串favorite的前六个字符添加到字符数组dest的末端，其效果与直接调用strcat函数相同，然后继续调用两次strncat函数，依次将字符串tabs和字符串language的首字符陆续连接到字符数组dest的末端，从而完成整个字符串子串的连接操作。

（3）最后将最终的结果输出，由于未将字符串tabs中的换行符添加到字符数组dest中，因此所有输出结果应在同一行。

本例程的运行结果是：

I love C

注意：本例程中，字符串tabs中的内容比较新奇，它并不是我们一般的字符，而是两个转义说明符构成的特殊字符，C语言内部在处理过程中遇到转义说明符时会作特殊处理，本例中会将' \t'看做制表符，将' \n'看做换行符。

# strncmp：字符串子串比较函数

函数原型： int strncmp (char \*str1,char \* str2, int n);

头文件：#include<string.h>

是否是标准函数：是

函数功能：比较两个字符串子串的大小，也就是把字符串str1的前n个字符组成的子串和字符串str2的前n个字符组成的子串进行比较，从首字符开始逐字符的进行比较，直到某个字符不相同或比较到第n个字符为止。

返回值：若字符串str1前n个字符组成的子串大于字符串str2前n个字符组成的子串返回结果大于零，若字符串str1前n个字符组成的子串小于字符串str2前n个字符组成的子串返回结果小于零，若字符串str1前n个字符组成的子串等于字符串str2前n个字符组成的子串返回结果等于零

例程如下： 应用strncmp比较字符串子串大小。

#include <string.h>

#include <string.h>

int main(void)

{

char \*str1="Hello World";

char \*str2="Hello C Programme";

int result;

result=strncmp(str1,str2,5);

if(!result)

printf("%s is identical to %s in the first 5 words",str1,str2);

else if(result<0)

printf("%s is less than %s in the first 5 words",str1,str2);

else

printf("%s is great than %s in the first 5 words",str1,str2);

printf("\n");

result=strncmp(str1,str2,10);

if(!result)

printf("%s is identical to %s in the first 10 words",str1,str2);

else if(result<0)

printf("%s is less than %s in the first 10 words",str1,str2);

else

printf("%s is great than %s in the first 10 words",str1,str2);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了两个字符串变量并分别赋予了初值，整型变量result用于记录字符串子串的比较结果。

（2）程序通过调用strncmp函数比较字符串str1和字符串str2的前5个字符组成的子串，由于两个字符串的前五个字符相同，因此两个子串的比较结果应为相等，返回结果为零。然后将比较结果输出。

（3）程序第二次调用strncmp函数比较字符串str2和字符串str3的前10个字符组成的子串，由于从第七个字符开始出现不等的情况，分别为' w'和' C'，根据ASIC码表中小写字母在后，即' w'的ASIC码大，返回结果大于零。最后输出比较结果。

（4）输出时显示的输出比较结果并指明比较范围。

本例程的运行结果是：

Hello World is identical to Hello C Programme in the first 5 words

Hello World is great than Hello C Programme in the first 10 words

注意：本例程中，要注意子串比较的过程中子串的大小应不小于零且不超过字符串的长度，虽然子串的长短参数不会产生编译时的错误和最终结果的输出，但在比较前检查比较范围是一个很好的习惯。

# strncpy：字符串子串拷贝函数

函数原型： char \* strncpy (char \*dest,char \* src, int n);

头文件：#include<string.h>

是否是标准函数：是

函数功能：实现字符串子串的拷贝工作，也就是把字符串src中的前n个字符拷贝到字符串dest中。

返回值：指向字符串dest的指针

例程如下： 应用strncpy实现字符串子串拷贝。

#include <stdio.h>

#include <string.h>

int main(void)

{

char dest[20]={""};

char \*src1="Hello World",\*src2 ="Aloha";

strncpy(dest,src1,5);

strncpy(dest,src2,5);

if(!strcmp(dest,src1))

printf("dest is equal to src1");

else if(!strcmp(dest,src2))

printf("dest is equal to src2");

else

printf("dest is %s",dest);

printf("%s\n", dest);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了两个字符串和一个字符数组并分别赋予了初值，本例中省去了用于记录比较结果result变量。

（2）程序通过调用strncpy函数将字符串src1中的前五个字符组成的子串拷贝到字符数组dest中，然后又将调用strncpy函数将字符串src2中的前五个字符组成的子串拷贝到字符数组dest中。通过调用一系列的strcmp字符串比较函数，从而达到验证dest变量中的最终内容的目的。

（3）最终的字符串dest中内容的到底是什么呢，是"Hello"，还是"Aloha"，亦或是" HelloAloha"。通过第一次调用strncpy函数，字符串dest中的内容由空串变成"Hello"，再次调用strncpy函数则会从字符串dest的下标为零处逐一覆盖，也就是"Aloha"覆盖了原来的" Hello "，并不是将"Aloha"添加到末端。

（4）最后将拷贝结果和验证结果输出。

本例程的运行结果是：

Aloha

dest is equal to src2

注意：本例程中，在检验字符串dest的内容时，if判断中并没有使用像上例中的result变量，我们只关心比较的结果是否为零，直接通过将函数作为判断条件，从而利用函数的返回值进行判断是一个简单而有效的方法。

# strpbrk：字符集字符匹配函数

函数原型：char \*strpbrk(char \*str1, char \*str2);

头文件：#include<string.h>

是否是标准函数：是

函数功能：在字符串中查找第一个属于字符集的字符位置，也就是在字符串str1中查找第一个属于字符集str2中任意字符的位置。

返回值：返回第一个匹配字符的指针

例程如下： 应用strpbrk匹配字符集字符。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*str1="There are 5 pigs in the hogpen";

char \*str2="0123456789";

char \*result;

result = strpbrk(str1,str2);

if(result)

printf("%s\n",result++);

else

printf("There are no numbers any more");

result = strpbrk(result,str2);

if(result)

printf("%s\n",result++);

else

printf("There are no numbers any more");

getch();

return 0;

}

例程说明：

（1）首先，程序声明了三个字符串变量并给前两个变量赋予初值，其中字符指针result用于记录匹配字符的位置。

（2）程序通过调用strcspn进行字符集字符的匹配，它从字符串str1中查找第一个属于字符集str2中任意字符的字符，具体到本例中就是在字符串str1种查找第一个数字字符，如果匹配成功我们会获得指向第一个数字字符的指针，利用该返回值输出匹配结果。

（3）然后我们在上一次匹配字符的下一个字符作为首字符的子串中继续匹配，程序第二次通过调用strspn完成上述功能，并用同样的输出方式输出匹配结果，如没有数字字符可以获得显示匹配失败。

（4）输出匹配结果时我们并没有将匹配的字符输出，取而代之的是将以匹配字符作为第一个字符的字符串字串输出。

本例程的运行结果是：

5 pigs in the hogpen

There are no numbers any more

注意：本例程中，值得注意的是匹配成功时结果的输出。由于获得了匹配成功的字符的指针，因此我们可以利用该指针输出字符串的字串，利用自增操作符我们移动一个位置又可以对尚未匹配的子串继续进行下一次匹配。

# strrchr：字符串中字符末次匹配函数

函数原型：char \*strrchr(char \*str, char c);

头文件：#include<string.h>

是否是标准函数：是

函数功能：在字符串中查找给定字符的最后一次匹配，也就是在字符串str中查找字符c最后一次出现的位置

返回值：最后一次匹配位置的指针

例程如下： 应用strrchr匹配字符串中字符。

#include <stdio.h>

#include <string.h>

int main(void)

{

char str[15]={""};

char \*ptr, c = 'o';

strcpy(str, "Hello World");

ptr = strchr(str, c);

if (ptr)

printf("The first character %c is at position: %d\n", c, ptr-str);

else

printf("The character was not found\n");

ptr = strrchr(str, c);

if (ptr)

printf("The last character %c is at position: %d\n", c, ptr-str);

else

printf("The character was not found\n");

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串和一个字符数组以及一个字符，并对字符变量赋予了我们要查找的值。

（2）程序通过调用strcpy 赋予了字符数组一个值，然后调用strchr函数在字符数组中查找第一次与字符变量c匹配的字符，也就是查找最后一个'o' 字符，返回的结果为指向第一个匹配字符的指针。根据返回值输出匹配结果。

（3）然后程序调用strrchr函数在字符数组中查找最后一次与字符变量c匹配的字符，也就是查找最后一个'o' 字符，最后根据返回值输出匹配结果。

（4）在字符数组中有两个'o'字符，因此调用strchr函数应该返回第一个'o'字符的指针，调用strrchr函数应该返回最后一个'o'字符的指针。

（5）对结果进行输出时，如果匹配成功，那么我们输出匹配的字符在数组中的位置，如果匹配不成功，则显示没有找到。

本例程的运行结果是：

The first character r is at position 4

The last character r is at position 7

注意：本例程中，如果字符串中只有一个'o'字符，那么无论调用哪种字符串中字符匹配函数都会返回相同的结果。

# strrev：字符串倒转函数

函数原型：char \*strrev(char \*str);

头文件：#include<string.h>

是否是标准函数：否

函数功能：将字符串进行倒转，也就是将字符串str中的第一个字符与最后一个字符交换，第二个字符与倒数第二个字符交换，以此类推。

返回值：返回倒转后字符串的指针

例程如下： 应用strrev将字符串倒转。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*str = "Able was I ere I saw Elba";

printf("Before: %s\n",str);

strrev(str);

printf("After: %s\n",str);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符数组并赋予初值，应该注意到字符数组的初值很有意思，类似于回文。

（2）程序通过调用strrev将原字符串中的所有内容倒转，第一个字符与最后一个字符交换，第二个字符与倒数第二个字符交换，以此类推。

（3）最后将倒转前后字符串的值打印出来。

本例程的运行结果是：

Able was I ere I saw Elba

ablE was I ere I saw elbA

注意：本例程中，字符数组中的初值并不是严格意义上的回文，将它倒转后会发现与原字符串并不是完全一样。

# strset：字符串设定函数

函数原型：char \*strset(char \*str, char c);

头文件：#include<string.h>

是否是标准函数：否

函数功能：将字符串原有字符全部设定为指定字符，也就是将字符串str中的所有字符全部用字符c进行替换.

返回值：返回指向被替换字符串的指针

例程如下： 应用strset将字符串设定为指定字符。

#include <stdio.h>

#include <string.h>

int main(void)

{

char str[11]="0123456789";

char symbol='a';

printf("Before: %s\n",str);

strset(str,symbol);

printf("After: %s\n",str);

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符数组和一个字符变量并赋予初值，字符变量代表用于替换的字符。

（2）程序通过调用strset将原字符串中的所有内容用字符'a'替换，相当于覆盖了原值并重新设定了字符串的值。

（3）最后将设定前后字符串的值打印出来。

本例程的运行结果是：

0123456789

aaaaaaaaaa

注意：本例程中，字符数组中指存储了十个字符，但是下表确是十一，利用字符串的相关知识可以理解该问题，有兴趣的读者可以将下表改成十或在字符串赋值的时候多加一个字符，看看程序会输出什么。

# strspn：字符集匹配函数

函数原型：int strspn(char \*str1, char \*str2);

头文件：#include<string.h>

是否是标准函数：是

函数功能：在字符串中查找第一个不属于字符集的下标，即从开始有多少个字符属于字符集，也就是在字符串str1中查找第一个不属于字符集str2中任何一个字符的下标，即字符串str1中从开始一直有多少个字符属于字符集str2中的字符。

返回值：所找到的字符串中段的长度

例程如下： 应用strspn匹配字符串中字符集。

#include <string.h>

#include <stdio.h>

Int main(void)

{

char \*str1="cabbage",\*str2="potato";

char \*str= "abc";

int result;

result = strspn(str1,str);

if(result)

printf("The first %d is congruent\n",result);

else

printf("No character is congruent");

result = strspn(str2,str);

if(result)

printf("The first %d is congruent\n",result);

else

printf("No character is congruent");

getch();

return 0;

}

例程说明：

（1）首先，程序声明了三个字符串并分别赋予初值，其中最后一个变量是用于匹配的字符集。

（2）程序通过调用strspn进行字符集的匹配，它从字符串str1中的第一个字符开始检查是不是属于字符串str中的任意字符，如果属于就继续匹配，直到匹配不成功，本例中会发现直到遇到字符'g'才匹配失败，因为字符'g'不属于字符串str中的任何一个字符。然后输出匹配结果。

（3）程序第二次通过调用strspn对字符串str2进行字符集匹配，发现第一个字符就不属于字符集str中的任意字符。

（4）输出匹配结果是显示前多少个字符匹配成功。

本例程的运行结果是：

The first 5 is congruent

No character is congruent

注意：本例程中，进行字符集匹配时，待匹配的字符串中的字符只要是字符集中的任意字符就匹配成功，要明确区分其余字符串匹配的不同。

# strstr：字符串匹配函数

函数原型：char \*strstr(char \*str1, char \*str2);

头文件：#include<string.h>

是否是标准函数：是

函数功能：在字符串中查找另一个字符串首次出现的位置，也就是在字符串str1中查找第一次出现字符串str2的位置。

返回值：返回第一次匹配字符串的指针

例程如下： 应用strstr匹配字符串。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*str1 = "Borland International",\*str2 = "nation";

char \*result;

result=strstr(str1, str2);

if(result)

printf("The substring is: %s\n", ptr);

else

printf("Not found the substring");

getch();

return 0;

}

例程说明：

（1）首先，程序声明了三个字符串变量并给前两个变量赋予初值，其中字符指针result用于记录匹配字符串的位置。

（2）程序通过调用strstr进行字符串匹配，查找字符串str1中首次出现字符串str2的位置，返回匹配结果。

（3）输出匹配结果时以匹配字符串的首字符作为子串的第一个字符输出，如果匹配不成功显示没有找到。

本例程的运行结果是：

The substring is national

注意：本例程中，匹配成功时的返回结果并不是进行匹配的字符串，而是第一次匹配成功的字符串首字符的指针。

# strtod：字符串转换成双精度函数

函数原型：double strtod(char \*str, char \*\*endptr);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：将字符串转换非双精度值，也就是将字符串str转换为双精度值，其中进行转换字符串必须是双精度数的字符表示格式，如果字符串中有非法的非数字字符，则第二个参数将负责获取该非法字符，即字符串指针endptr用于进行错误检测，转换在此非法字符处停止进行。

返回值：返回转换后的双精度结果

例程如下： 应用strtod将字符串转换为双精度值。

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main(void)

{

char str[20], \*endptr;

double result;

while(1)

{

printf("Input a float:");

gets(str);

result=strtod(str,&endptr);

if(result==-1)

printf("The number is %lf\n",str,result);

else

break;

}

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个用于转换的字符数组和一个用于进行错误检测的字符串指针，双精度result用于获取转换结果。

（2）程序通过循环不断接收从标准输入流中输入的字符串，并通过调用strtod将输入的字符串转换为双精度值，通过返回值获取转换结果，并通过第二个参数进行错误检测。循环的最后将转换结果打印出来。

（3）程序规定将字符串"-1"作为循环结束的标志，除非通过输入结束标志，否则循环条件总是成立的。

（4）如果输入一个正确的双精度字符串，程序会正确的转换并补齐尾数；如果输入整数，程序会自动将其转换成双精度数；如果小数点后面的位数过长，超过了双精度的范围，程序会采取截断的方式；如果输入期间出现了非法字符，程序停止转换并保留已转换的结果；如果第一个就是非法字符则返回零。

本例程的运行结果是：

Input a float: 4.2

The number is 4.20000

Input a float: 79

The number is 79.00000

Input a float: 1.1111111111

The number is 1.111111

Input a float: 34.45abc

The number is 34.450000

Input a float:abc

The number is 0.000000

Input a float: -1

注意：本例程中，即便转换出现非法字符循环也不会停止，而只是通过第二个参数捕捉到了非法字符，可以编写程序对非法字符进行处理，本例中并没有这样做，循环只是以输入循环结束标志循环结束依据。

# strtok：字符串分隔函数

函数原型：char \*strstr(char \*str1, char \*str2);

头文件：#include<string.h>

是否是标准函数：是

函数功能：在字符串中查找单词，这个单词始有第二个字符串中定义的分隔符分开，也就是在字符串str1中查找由字符串str2定义的分隔符，以分隔符为界，分隔出来的分隔符前面的所有字符组成一个单词，分离出第一个单词后将第一个参数置为空，可以继续分隔第二个单词。

返回值：返回分隔出的单词的指针

例程如下： 应用strtok分隔字符串。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*str1="I am very\thappy,to,stduy\nC\nprogramme";

char \*str2=" ,\t\n";

char \*token;

printf("%s\n\nTokens:\n",str1);

token = strtok(str1,str2);

while( token != NULL )

{

printf("%s\n",token);

token = strtok(NULL,str2);

}

getch();

return 0;

}

例程说明：

（1）首先，程序声明了三个字符串变量并给前两个变量赋予初值，其中第二个字符串是用于分隔的分隔符集，最后一个token用于记录分隔出来的单词，应该注意到第一个字符串中有许多分隔符。

（2）为了突出分隔结果，我们首先将字符串str1打印出来，它是按照标准输出的格式进行输出。

（3）程序通过调用strtok进行字符串分隔，查找字符串str1中首次出现字符串str2任意分隔符的位置，然后将分隔符之前的所有字符组成一个单词返回给token变量，从而得到分隔出来的首个单词。

（4）在循环体中，我们每次循环都要将上一次分隔出来的单词打印出来，另外就像前面所说，将strtok函数第一个参数置为空可以达到继续进行分隔的目的，也就是在上一次分隔出来的单词之后继续进行分隔，直到所有单词都分隔完毕，token变量会得到空的返回值，我们结束循环。

（5）打印分隔结果时，以换行区分每次分隔出的单词。

本例程的运行结果是：

I am very happy,to,study

C

Programme

Token:

I

am

very

happy

to

study

C

Programme

注意：本例程中，一定要记住如果在第一次分隔出单词后想继续进行分隔操作，务必要将函数的第一个参数置为空。

# strtol：字符串转换成长整型函数

函数原型：long strtol(char \*str, char \*\*endptr, int base);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：将字符串转换为长整型值，也就是将字符串str转换为长整型值，其中进行转换字符串必须是长整型的字符表示格式，如果字符串中有非法的非数字字符，则第二个参数将负责获取该非法字符，即字符串指针endptr用于进行错误检测，转换在此非法字符处停止进行。

返回值：返回转换后的长整型结果

例程如下： 应用strtol将字符串转换为长整型值。

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main(void)

{

char str[20], \*endptr;

long result;

while(1)

{

printf("Input a long:");

gets(str);

result=strtod(str,&endptr);

if(result!=-1)

printf("The number is %ld\n",result);

else

break;

}

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个用于转换的字符数组和一个用于进行错误检测的字符串指针，长整型result用于获取转换结果。

（2）程序通过循环不断接收从标准输入流中输入的字符串，并通过调用strtol将输入的字符串转换为长整型值，通过返回值获取转换结果，并通过第二个参数进行错误检测。循环的最后将转换结果打印出来。

（3）程序规定将字符串"-1"作为循环结束的标志，除非通过输入结束标志，否则循环条件总是成立的。

（4）如果输入一个正确的长整型字符串，程序会正确转换；如果输入小数，程序会将小数点后面的截断；如果转换的数过大，超过了长整型范围，程序返回长整型所能接受的最大值；如果输入期间出现了非法字符，程序停止转换并保留已转换的结果；如果第一个就是非法字符则返回零。

本例程的运行结果是：

Input a long: -15

The number is -15

Input a long: 1234.5678

The number is 1234

Input a long: 333333333333

The number is 2147483647

Input a long: -34abc

The number is -34

Input a long: abc

The number is 0

Input a float: -1

注意：本例程中，将字符串中的小数转换为长整型时，程序会将小数点看作非法字符，从而停止转换继续进行，因此无论小数点后面的数是多少都会截断，而不是我们习惯上的四舍五入或者五舍六入。

# strtoul：字符串转换成无符号长整型函数

函数原型：unsigned long strtoul(char \*str, char \*\*endptr, int base);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：将字符串转换为无符号长整型值，也就是将字符串str转换为无符号长整型值，其中进行转换字符串必须是无符号长整型的字符表示格式，如果字符串中有非法的非数字字符，则第二个参数将负责获取该非法字符，即字符串指针endptr用于进行错误检测，转换在此非法字符处停止进行。

返回值：返回转换后的无符号长整型结果

例程如下： 应用strtoul将字符串转换为无符号长整型值。

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main(void)

{

char str[20], \*endptr;

unsigned long result;

while(1)

{

printf("Input an unsigned long:");

gets(str);

result=strtoud(str,&endptr);

if(result!=-1)

printf("The number is %lu\n",result);

else

break;

}

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个用于转换的字符数组和一个用于进行错误检测的字符串指针，无符号长整型result用于获取转换结果。

（2）程序通过循环不断接收从标准输入流中输入的字符串，并通过调用strtoul将输入的字符串转换为无符号长整型值，通过返回值获取转换结果，并通过第二个参数进行错误检测。循环的最后将转换结果打印出来。

（3）程序规定将字符串"1"作为循环结束的标志，除非通过输入结束标志，否则循环条件总是成立的。

（4）如果输入一个正确的长整型字符串，程序会正确转换；如果输入一个负数，程序返回零。

本例程的运行结果是：

Input a long: 100

The number is 100

Input a long: -36

The number is 0

Input a float: 1

注意：本例程中，输入负数的时候程序会将负号看作非法字符，从而停止转换继续进行，没有发生任何实际的转换。

# strupr：字符串大写转换函数

函数原型：char \*strupr(char \*str);

头文件：#include<string.h>

是否是标准函数：否

函数功能：将字符串原有小写字符全部转换为大写字符，也就是将字符串str中的所有字符变成大写。

返回值：返回指向被转换字符串的指针

例程如下： 应用strupr将字符串转换成大写字符。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*s=" You'll Never Walk Alone ";

printf("%s",strlwr(s));

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串为待转换字符串并赋予初值。

（2）程序通过调用strupr将字符串中的所有小写字符转换成大写字符，并返回转换后的结果。

（3）最后将转换结果打印出来。

本例程的运行结果是：

YOU’LL NEVER WALK ALONE

# strupr：字符串大写转换函数

函数原型：char \*strupr(char \*str);

头文件：#include<string.h>

是否是标准函数：否

函数功能：将字符串原有小写字符全部转换为大写字符，也就是将字符串str中的所有字符变成大写。

返回值：返回指向被转换字符串的指针

例程如下： 应用strupr将字符串转换成大写字符。

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*s=" You'll Never Walk Alone ";

printf("%s",strlwr(s));

getch();

return 0;

}

例程说明：

（1）首先，程序声明了一个字符串为待转换字符串并赋予初值。

（2）程序通过调用strupr将字符串中的所有小写字符转换成大写字符，并返回转换后的结果。

（3）最后将转换结果打印出来。

本例程的运行结果是：

YOU’LL NEVER WALK ALONE

## 第五章（数学函数）

[1. abs、labs、fabs：求绝对值函数 1](#_Toc278280919)

[2. acos： 反余弦函数 2](#_Toc278280920)

[3. asin： 反正弦函数 3](#_Toc278280921)

[4. atan：反正切函数 3](#_Toc278280922)

[5. atan2：反正切函数2 4](#_Toc278280923)

[6. ceil：向上舍入函数 4](#_Toc278280924)

[7. cos ：余弦函数 5](#_Toc278280925)

[8. cosh： 双曲余弦函数 5](#_Toc278280926)

[9. div、ldiv：除法函数 6](#_Toc278280927)

[10. exp：求e的x次幂函数 7](#_Toc278280928)

[11. floor：向下舍入函数 7](#_Toc278280929)

[12. fmod：求模函数 8](#_Toc278280930)

[13. frexp：分解浮点数函数 8](#_Toc278280931)

[14. hypot：求直角三角形斜边长函数 9](#_Toc278280932)

[15. ldexp：装载浮点数函数 10](#_Toc278280933)

[16. log、log10：对数函数 10](#_Toc278280934)

[17. modf：分解双精度数函数 11](#_Toc278280935)

[18. pow、pow10：指数函数 11](#_Toc278280936)

[19. rand：产生随机整数函数 12](#_Toc278280937)

[20. sin：正弦函数 13](#_Toc278280938)

[21. sinh： 双曲正弦函数 13](#_Toc278280939)

[22. sqrt：开平方函数 14](#_Toc278280940)

[23. srand：设置随机时间的种子函数 14](#_Toc278280941)

[24. tan： 正切函数 15](#_Toc278280942)

[25. tanh：双曲正切函数 15](#_Toc278280943)

# abs、labs、fabs：求绝对值函数

函数原型：int abs(int x);

long labs(long x);

double fabs(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：函数int abs(int x);是求整数x的绝对值；函数long labs(long n);是求长整型数x的绝对值；函数double fabs(double x); 是求浮点数x的绝对值。

返回值：返回计算结果。

例程如下： 计算整数的绝对值。

#include <math.h>

int main(void)

{

int x = -56;

printf("number: %d absolute value: %d\n",x, abs(x));

return 0;

}

例程说明：

本例程通过abs函数计算出整型数-56的绝对值56，并在屏幕上显示结果。本例程的运行结果为：

number: -56 absolute value: 56

例程如下：计算长整数的绝对值。

#include <math.h>

int main(void)

{

long x = -12345678L;

printf("number: %ld absolute value: %ld\n", x,labs(x));

return 0;

}

例程说明：

本例程通过labs函数计算出长整型数-12345678的绝对值12345678，并在屏幕上显示结果。本例程的运行结果为：

number: -12345678 absolute value: 12345678

例程如下：计算浮点数的绝对值。

#include <math.h>

int main(void)

{

float x = -128.0;

printf("number: %f absolute value: %f\n", x, fabs(x));

return 0;

}

例程说明：

本例程通过fabs函数计算出浮点数-128.0的绝对值128.0，并在屏幕上显示结果。本例程的运行结果为：

number: -128.000000 absolute value: 128.000000

# acos： 反余弦函数

函数原型：double acos(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求x的反余弦值，这里，x为弧度，x的定义域为[-1.0，1.0]，arccosx的值域为[0，π]。

返回值：计算结果的双精度值。

例程如下： 求arccosx。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = 1.0;

result = acos(x);

printf("arccos %lf=%lf\n", x, result);

return 0;

}

例程说明：

本例程应用函数acos计算1.0的反余弦值，即：arccos1。返回计算结果的双精度值。本例程的运行结果是：

arccos 1.000000=0.000000

# asin： 反正弦函数

函数原型：double asin(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求x的反正弦值，这里，x为弧度，x的定义域为[-1.0，1.0]，arcsinx值域为[-π/2，+π/2]。

返回值：计算结果的双精度值。

例程如下：求arcsinx。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = 1.0;

result = asin(x);

printf("arcsin %lf is %lf\n", x, result);

return(0);

}

例程说明：

本例程应用函数asin计算1.0的反正弦值，即：arcsin1。返回计算结果的双精度值。本例程的运行结果是：

arcsin 1.000000 is 1.570796

# atan：反正切函数

函数原型：double atan(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求x的反正切值，这里，x为弧度，x的定义域为(-∞,+∞)，arctanx的值域为(-π/2，+π/2)。

返回值：计算结果的双精度值。

例程如下： 求arctanx。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = 1.0;

result = atan(x);

printf("arctan %lf = %lf\n", x, result);

return(0);

}

例程说明：

本例程应用函数atan计算1.0的反正切值，即：arctan1。并返回计算结果的双精度值。本例程的运行结果是：

arctan 1.000000 = 0.785398

# atan2：反正切函数2

函数原型：double atan2(double y, double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求y/x的反正切值。

返回值：计算结果的双精度值。

例程如下： 求arctan(x/y)。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = 10.0, y = 5.0;

result = atan2(y, x);

printf("arctan%lf = %lf\n", (y / x), result);

return 0;

}

例程说明：

本例程应用函数atan2计算10.0/5.0的反正切值，即：arctan0.5。并返回计算结果的双精度值。本例程的运行结果是：

arctan0.500000 = 0.463648

# ceil：向上舍入函数

函数原型：double ceil(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：将双精度数x向上舍入，即：取它的最大整数。例如：ceil(123.400000)=124.000000。

返回值：返回计算结果。

例程如下： 数值的向上舍入。

#include <math.h>

#include <stdio.h>

int main(void)

{

double num = 123.400000;

double up;

up = ceil(num);

printf("The original number %lf\n", num);

printf("The num rounded up %lf\n", up);

return 0;

}

例程说明：

本例程通过函数ceil将双精度数123.400000向上舍入，得到的结果为124.000000，并在屏幕上显示运算结果。本例程的运行结果为：

The original number 123.400000

The num rounded up 124.000000

# cos ：余弦函数

函数原型：double cos(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求x的余弦值，这里，x为弧度。

返回值：计算结果的双精度值。

例程如下： 求cosx。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = M\_PI;

result = cos(x);

printf("cos(PI) is %lf\n", result);

return 0;

}

例程说明：

本例程应用cos函数计算π的余弦值，即：cosπ，再返回计算结果的双精度值。本例程的运行结果是：

cos(PI) is -1.000000

# cosh： 双曲余弦函数

函数原型：dluble cosh(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：计算x的双曲余弦值。其中ch(x)=(ex+e-x)/2。

返回值：计算结果的双精度值。

例程如下： 求x的双曲余弦值ch(x)。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = 0.5;

result = cosh(x);

printf("ch(%lf) = %lf\n", x, result);

return 0;

}

例程说明：

本例程应用函数cosh计算0.5的双曲余弦值，即：ch(0.5)，并返回计算结果的双精度值。本例程的运行结果是：

ch(0.500000) = 1.127626

# div、ldiv：除法函数

函数原型：div\_t div(int number, int denom);

ldiv\_t ldiv(long lnumer, long ldenom);

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：函数div是将两个整数numbe和denom相除, 返回商和余数。函数ldiv是将两个长整数lnumbe和ldenom相除, 返回商和余数。

返回值：函数div返回div\_t类型的值；函数ldiv返回ldiv\_t类型的值。

例程如下：两整数相除，求其商和余数。

#include <stdlib.h>

#include <stdio.h>

div\_t x;

int main(void)

{

x = div(11,5);

printf("11 div 5 = %d remainder %d\n", x.quot, x.rem);

return 0;

}

例程说明：

本例程通过div函数将11和5相除，返回其商和余数。

注意：div函数并不是<math.h>中的函数，而是<stdlib.h>中的函数。<stdlib.h>中包含存储分配函数和一些杂项函数。但由于div函数具有数学计算的功能，因此将其归类到数学函数中。

div\_t是<stdlib.h>中定义的数据类型，它是一个结构体，定义如下：

typedef struct

{

int quot; /\*商\*/

int rem; /\*余数\*/

}div\_t;

其中包含两个域：商和余数。div函数将两个整数相除, 返回一个div\_t类型的值。

该函数的运行结果是：

11 div 2 = 5 remainder 1

例程如下：两长整数相除，求其商和余数。

#include <stdlib.h>

#include <stdio.h>

ldiv\_t lx;

int main(void)

{

lx = ldiv(200000L, 70000L);

printf("200000 div 70000 = %ld remainder %ld\n", lx.quot, lx.rem);

return 0;

}

例程说明：

本例程通过ldiv函数将长整数200000与70000相除，并返回其商和余数。

注意：同函数div一样，函数ldiv是<stdlib.h>中的函数。

ldiv\_t是<stdlib.h>中定义的数据类型，它是一个结构体，定义如下：

typedef struct {

long quot;

long rem;

} ldiv\_t;

其中包含两个域：商和余数。ldiv函数将两个长整数相除, 返回一个ldiv\_t类型的值。

该函数的运行结果是：

200000 div 70000 = 2 remainder 60000

# exp：求e的x次幂函数

函数原型：double exp(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：计算自然常数e的x幂。

返回值：返回计算结果的双精度值。

例程如下： 计算ex（说明：e=2.718281828...）。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = 3.0;

result = exp(x);

printf("'e' raised to the power of %lf (e ^ %lf) = %lf\n", x, x, result);

return 0;

}

例程说明：

本例程应用函数exp计算e3，该函数返回计算结果的双精度值。本例程的运行结果为：

'e' raised to the power of 3.000000 (e ^ 3.000000) = 20.085537

# floor：向下舍入函数

函数原型：double floor(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：将双精度数x向下舍入，即：取它的最小整数。例如：floor(123.400000)=123.000000。

返回值：返回计算结果。

例程如下：数值的向下舍入。

#include <math.h>

#include <stdio.h>

int main(void)

{

double num = 123.400000;

double up;

up = floor(num);

printf("The original number %lf\n", num);

printf("The num rounded down %lf\n", up);

return 0;

}

例程说明：

本例程通过函数floor将双精度数123.400000向下舍入，得到的结果为123.000000，并在屏幕上显示运算结果。本例程的运行结果为：

The original number 123.400000

The num rounded down 123.000000

# fmod：求模函数

函数原型：double fmod(double x, double y);

头文件：#include<math.h>

是否是标准函数：是

函数功能：计算x对y的模, 即x/y的余数。

返回值：返回计算结果，即余数的双精度。

例程如下： 计算两数的余数。

#include <stdio.h>

#include <math.h>

int main(void)

{

float x,y;

x=12.580000;

y=2.600000;

printf("12.580000/2.600000: %f\n",fmod(x,y));

getchar();

return 0;

}

例程说明：

本例程通过函数fmod求双精度数12.580000和2.600000的模，其结果为：2.180000。

本例程的运行结果为：

12.580000/2.600000: 2.180000

# frexp：分解浮点数函数

函数原型：double frexp(double val, int \*exp);

头文件：#include<math.h>

是否是标准函数：是

函数功能：把浮点数或双精度数val分解为数字部分（尾数部分）x和以2为底的指数部分n。即val=x\*2n，其中n存放在exp指向的变量中。

返回值：返回尾数部分x的双精度值，且0.5≤x<1。

例程如下： 应用函数frexp分解浮点数。

#include <stdio.h>

#include <math.h>

int main(void)

{

float x;

int exp;

x=frexp(64.0,&exp);

printf("64.0=%.2f\*2^%d",x,exp);

getchar();

return 0;

}

例程说明：

本例程通过函数frexp将浮点数64.0分解为尾数0.5和以2为底的指数7。该函数将指数7存放在变量exp 中，并返回一个双精度的尾数0.500000。本例程的运行结果为：

64.0=0.50\*2^7

# hypot：求直角三角形斜边长函数

函数原型：double hypot(double x, double y);

头文件：#include<math.h>

是否是标准函数：是

函数功能：x,y为给定的直角三角形两直角边，求该直角三角形的斜边。

返回值：返回计算结果的双精度值。

例程如下： 根据两直角边求斜边的长。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result;

double x = 3.0;

double y = 4.0;

result = hypot(x, y);

printf("The hypotenuse is: %lf\n", result);

return 0;

}

例程说明：

本例程中，已知两直角边长度：x = 3.0; y = 4.0，应用函数hypot求出其斜边长度。本例程的运行结果为：

The hypotenuse is: 5.000000

# ldexp：装载浮点数函数

函数原型：double ldexp(double val, int exp);

头文件：#include<math.h>

是否是标准函数：是

函数功能：功能与函数frexp相反，将给定的尾数，指数装载成相应的双精度数或浮点数。即计算val\*2n，其中n为参数exp 的值。

返回值：返回val\*2n的计算结果。

例程如下：应用函数frexp装载浮点数。

#include <stdio.h>

#include <math.h>

int main(void)

{

double value;

double x = 3.000000;

value = ldexp(x,3);

printf("The ldexp value is: %lf\n", value);

getchar();

return 0;

}

例程说明：

本例程通过函数ldexp将尾数3.000000与指数3装载成相应的双精度数。即：3.000000\*23= 24.000000。该函数返回一个双精度数。本例程的运行结果为：

The ldexp value is: 24.000000

# log、log10：对数函数

函数原型：double log(double x);

double log10(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求对数。函数log是求以e为底的x的对数（自然对数）即：lnx；函数log10是求以10为底的x的对数，即：log10x。

返回值：返回计算结果的双精度值。

例程如下： 计算lnx。

#include <math.h>

#include <stdio.h>

int main(void)

{

double result;

double x = M\_E;

result = log(x);

printf("The natural log of %lf is %lf\n", x, result);

return 0;

}

例程说明：

本例程应用函数log计算双精度数M\_E的自然对数，其中M\_E为<math.h>中定义的常数#define M\_E 2.71828182845904523536就等于e，因此，本例程的运行结果为：

The natural log of 2.718282 is 1.000000

例程如下： 计算log10x。

#include <math.h>

#include <stdio.h>

int main(void)

{

double result;

double x = 1000.0 ;

result = log10(x);

printf("The common log of %lf is %lf\n", x, result);

return 0;

}

例程说明：

本例程应用函数log10计算双精度数1000.0的以10为底的对数，该函数返回的结果仍是双精度数。本例程的运行结果为：

The common log of 1000.000000 is 3.000000

# modf：分解双精度数函数

函数原型：double modf(double num, double \*i);

头文件：#include<math.h>

是否是标准函数：是

函数功能：把双精度数num分解为整数部分和小数部分，并把整数部分存到i指向的单元中。

返回值：返回num的小数部分的双精度值。

例程如下： 应用函数modf分解双精度数。

#include <math.h>

#include <stdio.h>

int main(void)

{

double fraction, integer;

double number = 12345.6789;

fraction = modf(number, &integer);

printf("The integer and the fraction of %lf are %lf and %lf\n",

number, integer, fraction);

return 0;

}

例程说明：

本例程将双精度数12345.6789分解为整数部分和小数部分，并将整数部分存入变量integer中，返回小数部分。最后在屏幕上显示结果。本例程的运行结果为：

The integer and the fraction of 12345.678900 are 12345.000000 and 0.678900

# pow、pow10：指数函数

函数原型：double pow(double x, double y);

double pow10(int x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：指数函数。函数pow是求x的y次方；函数pow10相当于pow(10.0,x)，是求10的x次方。

返回值：返回计算结果的双精度值。

例程如下： 计算xy 。

#include <math.h>

#include <stdio.h>

int main(void)

{

double x = 2.0, y = 10.0;

printf("The result of %lf raised to %lf is %lf\n", x, y, pow(x, y));

return 0;

}

例程说明：

本例程中，应用函数pow计算210，并将结果的双精度值返回。本例程的运行结果为：

The result of 2.000000 raised to 10.000000 is 1024.000000

例程如下： 计算10x。

#include <math.h>

#include <stdio.h>

int main(void)

{

double x = 2.0;

printf("The result of 10 raised to %lf is %lf\n", x, pow10(x));

return 0;

}

本例程中，应用函数pow10计算102，并将结果的双精度值返回。本例程的运行结果为：

The result of 10 raised to 2.000000 is 100.000000

# rand：产生随机整数函数

函数原型：int rand(void);

头文件：#include<math.h>

是否是标准函数：是

函数功能：产生-90到32767之间的随机整数。

返回值：产生的随机整数。

例程如下： 利用函数rand产生处于0~99之间的5个随机整数。

#include <stdlib.h>

#include <stdio.h>

int main(void)

{

int i;

printf("Random numbers from 0 to 99\n");

for(i=0; i<5; i++)

printf("%d ", rand() % 100);

return 0;

}

例程说明：

循环地应用函数rand产生随机整数，并利用rand() % 100将范围控制在0~99之间。共产生5个0~99之间的随机整数。本例程的运行结果为：

Random numbers from 0 to 99

46 30 82 90 56

注意：rand不是<math.h>中定义的函数，而是<stdlib.h>中定义的函数。因此要在源程序中包含<stdlib.h>头文件。

# sin：正弦函数

函数原型：double sin(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求x的正弦值，这里，x为弧度。

返回值：计算结果的双精度值。

例程如下： 求sinx。

#include <stdio.h>

#include <math.h>

int main(void)

{

float x;

x=M\_PI/2;

printf("sin(PI/2)=%f",sin(x));

getchar();

return 0;

}

例程说明：

本例程应用sin函数计算π/2的正弦值，即：sin(π/2)。返回计算结果的双精度值。

注意：M\_PI是<math.h>中定义的π值常量。本例程的运行结果为：

sin(PI/2)=1.00000

# sinh： 双曲正弦函数

函数原型：double sinh(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：计算x的双曲正弦值。其中sh(x)=(ex-e-x)/2。

返回值：计算结果的双精度值。

例程如下： 求x的双曲正弦值sh(x)。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result, x = 0.5;

result = sinh(x);

printf("sh( %lf )=%lf\n", x, result);

return 0;

}

例程说明：

本例程应用函数sinh计算0.5的双曲正弦值，即：sh(0.5)，并返回计算结果的双精度值。本例程的运行结果是：

sh( 0.500000 )=0.521095

# sqrt：开平方函数

函数原型：double sqrt(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求双精度数x的算术平方根，这里，x≥0。

返回值：返回计算结果的双精度值。

例程如下： 计算双精度数的平方根。

#include <math.h>

#include <stdio.h>

int main(void)

{

double result,x = 4.0;

result = sqrt(x);

printf("The square root of %lf is %lf\n", x, result);

return 0;

}

例程说明：

本例程中，应用函数sqrt计算出4.0的平方根，并将结果的双精度值返回。本例程的运行结果为：

The square root of 4.000000 is 2.000000

# srand：设置随机时间的种子函数

函数原型：int srand (unsigned int seed);

头文件：#include<math.h>

是否是标准函数：是

函数功能：设置随机时间的种子，常与rand()结合使用。否则如果直接用rand函数产生随机数，每次运行程序的结果都相同。

返回值：

例程如下： 产生不同的随机整数序列。

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

int main(void)

{

int i;

time\_t t;

srand((unsigned) time(&t));

printf("Random numbers from 0 to 99\n");

for(i=0; i<5; i++)

printf("%d ", rand() % 100);

return 0;

}

例程说明：

（1）首先，程序应用函数time获取系统时间作为种子，并强制转换为unsigned型变量，作为函数srand的参数。

（2）再利用rand函数产生5个0~99之间的随机整数。这样，每次运行该程序，都以当时的系统时间作为随机时间的种子，产生的随机数就不会重复了。

注意：time\_t是<time.h >中定义的数据类型，用以描述时间。而函数time可以获取当前的系统时间。运行两次本例程，可得到两组不同的随机数序列：

Random numbers from 0 to 99

23 16 92 26 99

Random numbers from 0 to 99

60 72 77 40 45

# tan： 正切函数

函数原型：double tan(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求x的正切值，这里，x为弧度，其中x≠k(π/2)，k为整数。

返回值：计算结果的双精度值。

例程如下： 求tanx。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result, x;

x = M\_PI/4;

result = tan(x);

printf("tan (PI/4)=%lf\n", result);

return 0;

}

例程说明：

本例程应用tan函数计算π/4的正切值，即：tan(π/4)。并返回计算结果的双精度值。本例程的运行结果是：

tan (PI/4)=1.00000

# tanh：双曲正切函数

函数原型：double tanh(double x);

头文件：#include<math.h>

是否是标准函数：是

函数功能：求x的双曲正切值。其中th(x)=sh(x)/ch(x)= (ex-e-x)/ (ex+e-x)。

返回值：计算结果的双精度值。

例程如下： 求x的双曲正切值th(x)。

#include <stdio.h>

#include <math.h>

int main(void)

{

double result, x;

x = 0.5;

result = tanh(x);

printf("th(%lf) = %lf\n", x, result);

return 0;

}

例程说明：

本例程应用函数tanh计算0.5的双曲正切值，即：th(0.5)，并返回计算结果的双精度值。本例程的运行结果是：

th(0.500000) = 0.462117

## 第六章（时间和日期函数）

[1. asctime：日期和时间转换函数 1](#_Toc278278947)

[2. clock：测定运行时间函数 2](#_Toc278278948)

[3. ctime：时间转换函数 2](#_Toc278278949)

[4. difftime：计算时间差函数 3](#_Toc278278950)

[5. gmtime： 将日历时间转换为GMT 4](#_Toc278278951)

[6. localtime：把日期和时间转变为结构 5](#_Toc278278952)

[7. mktime：时间类型转换函数 5](#_Toc278278953)

[8. time：获取系统时间函数 7](#_Toc278278954)

# asctime：日期和时间转换函数

函数原型：char \*asctime(const struct tm \*tblock)

头文件：#include<time.h>

是否是标准函数：是

函数功能：本函数把指定的tm结构类的日期(分段日期)转换成下列格式的字符串:

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

返回值：转换后的字符串指针。

例程如下： 用asctime函数转换时间格式。

#include <stdio.h>

#include <string.h>

#include <time.h>

int main(void)

{

struct tm t;

char str[80];

/\*设置tm结构类变量t的时间成员 \*/

t.tm\_sec = 1; /\* 秒\*/

t.tm\_min = 30; /\*分钟 \*/

t.tm\_hour = 9; /\* 时 \*/

t.tm\_mday = 22; /\* 日 \*/

t.tm\_mon = 11; /\* 月 \*/

t.tm\_year = 56; /\* 年 \*/

t.tm\_wday = 4; /\*星期\*/

t.tm\_yday = 0; /\*不必设置\*/

t.tm\_isdst = 0; /\*不必设置 \*/

/\*格式转换 \*/

strcpy(str, asctime(&t));

printf("%s\n", str);

return 0;

}

例程说明：

（1）首先定义tm结构类的变量t，并设置t的时间成员。

（2）通过函数asctime将t的时间转换为指定格式。

（3）输出转换后的指定格式的字符串。

本例程的运行结果为：

Thu Dec 22 09:30:01 1956

注意：

函数asctime返回指向转换后的指定格式的字符串指针。本例程通过函数strcpy将指定格式的字符串的指针拷贝给str，并通过“%s”格式符输出该指定格式的字符串。

# clock：测定运行时间函数

函数原型：clock\_t clock(void)

头文件：#include<time.h>

是否是标准函数：是

函数功能：确定所用的处理器时间。函数clock返回实现环境中从程序运行开始所用的处理器时间的最佳近似值，仅与程序启动有关。clock函数无参数。

返回值：如果成功，返回从程序开始运行经过的时间；否则（系统没有内部时钟）返回-1。

例程如下： 应用clock函数计算程序运行时间

#include <time.h>

#include <stdio.h>

#include <dos.h>

int main(void)

{

clock\_t start, end;

/\*程序运行到现在的时间\*/

start = clock();

/\*间隔1000毫秒\*/

delay(1000);

/\*程序运行到现在的时间\*/

end = clock();

printf("The time was: %f\n", (end - start) / CLK\_TCK);

return 0;

}

例程说明：

（1）首先用clock函数记录下程序运行到目前所用的时间，并将该时间存入clock\_t类型的变量start中。

（2）应用delay函数使程序暂停1000毫秒。

（3）再用clock函数记录下程序运行到目前所用的时间，并将该时间存入clock\_t类型的变量end中。

（4）计算出时间差（end-start）获得程序暂停delay的时间。再除以常量CLK\_TCK，转化为以秒为单位。

本例程的执行结果为：

The time was:0.989011

注意：CLK\_TCK是系统常量。

# ctime：时间转换函数

函数原型：char \*ctime(const time\_t \*time)

头文件：#include<time.h>

是否是标准函数：是

函数功能：将time所指向的日历时间转换为字符串形式的本地时间。它等价于函数调用asctime(localtime(timer))。字符串的格式为：DDD MMM dd hh:mm:ss YYYY

返回值：转换后的字符串指针。

例程如下： 用 ctime函数转换时间格式。

#include <stdio.h>

#include <time.h>

int main(void)

{

time\_t t;

time(&t);

printf("Today's date and time: %s\n", ctime(&t));

return 0;

}

例程说明：

（1）首先定义time\_t类型的变量t。

（2）应用函数time获取系统时间。

（3）通过函数ctime 将获取的日历时间time\_t转换为规定格式的字符串表示。

本例程的运行结果为：

Today's date and time: Sat Nov 10 00:57:14 2007

注意：

函数ctime是将日历时间直接转换为规定格式的字符串表示：

DDD MMM dd hh:mm:ss YYYY

其中，“DDD”表示一星期中的某一天，例如“Sat”表示星期六；“MMM”表示月份，例如“Nov”就表示十一月；dd hh:mm:ss为时钟显示；YYYY为年份。

# difftime：计算时间差函数

函数原型：double difftime(time\_t time2, time\_t time1)

头文件：#include<time.h>

是否是标准函数：是

函数功能：计算两个日历时间time1和time2的时间间隔。其中time1为指定的第一个时间，time2为指定的第二个时间。time1要小于或等于time2。

返回值：返回时间差，以秒为单位的double类型。

例程如下： 应用函数difftime计算时间差

#include <time.h>

#include <stdio.h>

#include <dos.h>

#include <conio.h>

int main(void)

{

time\_t first, second;

clrscr();

/\*获得系统时间 \*/

first = time(NULL);

/\* 等待2秒 \*/

delay(2000);

/\* 再次获得系统时间 \*/

second = time(NULL);

printf("The Interval is: %f seconds\n",difftime(second,first));

getch();

return 0;

}

例程说明：

（1）首先通过time函数获得系统时间，并将其存储在time\_t类型变量first中。

（2）应用delay函数使程序暂停2000毫秒。

（3）再次通过time函数获得系统时间，并将其存储在time\_t类型变量second中。

（4）通过函数difftime获得first和second的时间间隔，并显示在屏幕上。

本例程的执行结果为：

The Interval is: 2.000000 seconds

# gmtime： 将日历时间转换为GMT

函数原型：struct tm \*gmtime(const time\_t \*timer)

头文件：#include<time.h>

是否是标准函数：是

函数功能：把日期和时间转换为格林尼治标准时间(GMT)。

返回值：指向struct tm分段日期结构类型的指针。

例程如下： 将日历时间转换为GMT。

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(void)

{

time\_t t;

struct tm \*gmt;

t=time(NULL);

gmt=gmtime(&t);

printf("GMT is:%s", asctime(gmt));

return 0;

}

例程说明：

（1）首先利用time函数获取系统时间，它是一个日历时间。

（2）再应用gmtime函数将该日历时间转换为格林尼治标准时间(GMT)。该函数返回一个指向struct tm分段日期结构类型的指针。

（3）最后应用asctime函数将分段日期转换成规定格式的字符串表示。

本例程的运行结果为：

GMT is:Sat Nov 10 06:25:13 2007

# localtime：把日期和时间转变为结构

函数原型：struct tm \*localtime(const time\_t \*timer)

头文件：#include<time.h>

是否是标准函数：是

函数功能：把timer所指的日历时间转换为以本地时间表示的分段时间。

返回值：指向struct tm分段日期结构类型的指针。

例程如下： 利用函数localtime将日历时间转换为分段时间。

#include <time.h>

#include <stdio.h>

#include <dos.h>

int main(void)

{

time\_t timer;

struct tm \*tblock;

timer = time(NULL);

tblock = localtime(&timer);

printf("Local time is: %s\n", asctime(tblock));

printf("Today's date and time: %s\n", ctime(&timer)) ;

getchar();

return 0;

}

例程说明：

（1）首先利用time函数获取系统时间，它是一个日历时间。

（2）利用函数localtime将获取的系统时间（它是一个日历时间）转换为分段时间tm。

（3）利用函数asctime将该分段时间转换为规定的字符串格式，并显示。

（4）利用函数ctime直接将日历时间转换为规定的字符串格式，并显示。

本例程的运行结果为：

Local time is: Sat Nov 10 01:47:59 2007

Today's date and time: Sat Nov 10 01:47:59 2007

注意：

前面讲过，函数调用asctime(localtime(&time))等价于函数调用ctime(&time)，其中&time为日历时间数据的指针。因此，本例程中的两种输出方式结果是一样的。

# mktime：时间类型转换函数

函数原型：time\_t mktime(struct tm\*timeptr)

头文件：#include<time.h>

是否是标准函数：是

函数功能：将tm类型的结构指针timeptr指向的结构体中的日期与时间转换为time\_t类型的日期和时间，并返回。

返回值：time\_t类型的日期和时间，如果日历不能被表达，返回-1。

例程如下： 输出指定日期是一周的哪一天。

#include <time.h>

#include <stdio.h>

#include <dos.h>

#include <conio.h>

int main(void)

{

char \*week\_day [8]={ "Sun",

"Mon",

"Tue",

"Wed",

"Fri",

"Sat",

"Unknow",

};

struct tm t;

/\*指定日期时间\*/

t.tm\_year=99;

t.tm\_mon=1;

t.tm\_mday=1;

t.tm\_hour=0;

t.tm\_min=0;

t.tm\_sec=1;

t.tm\_isdst=-1;

/\*调用函数mktime设置tm\_wday成员\*/

if(mktime(&t)==-1)

t.tm\_wday=7;

printf("The day is:%s",week\_day[t.tm\_wday]);

getch();

return 0;

}

例程说明：

（1）首先定义一个struct tm结构类型的变量t，并指定t 的日期与时间。

（2）通过函数mktime将t中指定的分段时间转化为日历时间。如果日历时间不能被表达，即返回-1，则将t.tm\_wday赋值为0。

（3）根据字符串数组week\_day的初值，显示t中指定的时间是一周中的哪一天。

注意：

1、struct tm是time.h中定义的结构体，用来指定分段日期与时间。tm结构如下：

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;

};

2、调用函数mktime的作用是将tm类型结构指针指定的日期和时间转换为time\_t类型的日历时间，该值与函数time返回值的编码方式相同。函数mktime调用成功时，成员tm\_wday被设置为适当的值，并返回一个time\_t类型的日历时间。因此本例程中，如果mktime(&t)==-1，则表示函数mktime执行不成功，则将t.tm\_wday置为7，输出"Unknow"。否则t. tm\_wday会被设置为适当的值。

# time：获取系统时间函数

函数原型：time\_t time(time\_t \*tp)

头文件：#include<time.h>

是否是标准函数：是

函数功能：获取系统时间。

返回值：time\_t类型的当前日历时间的最佳近似值，如果日历不能被表达，返回-1。

例程如下： 应用函数time获取系统时间。

#include <time.h>

#include <stdio.h>

int main(void)

{

time\_t t;

t = time(&t);

printf("The number of seconds since January 1, 1970 is %ld",t);

return 0;

}

例程说明：

（1）首先定义time\_t类型的变量t。

（2）通过函数time获取从GMT1970年1月1日00:00:00开始经过的秒数。

（3）输出该秒数。

本例程的运行结果为：

The number of seconds since January 1, 1970 is 1047682192

## 第七章（其它函数）

[1. abort：异常终止进程函数 1](#_Toc278281389)

[2. atexit：注册终止函数 2](#_Toc278281390)

[3. bsearch：二分搜索函数 2](#_Toc278281391)

[4. calloc：分配主存储器函数 4](#_Toc278281392)

[5. exit：正常终止进程函数 5](#_Toc278281393)

[6. free：释放内存函数 6](#_Toc278281394)

[7. getenv：获取环境变量 6](#_Toc278281395)

[8. malloc：动态分配内存函数 7](#_Toc278281396)

[9. qsort：快速排序函数 7](#_Toc278281397)

[10. realloc：重新分配主存函数 9](#_Toc278281398)

# abort：异常终止进程函数

函数原型：void abort(void)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：异常终止一个进程，并打印一条终止信息“Abnormal program termination”到stderr。

返回值：无返回值。

例程如下： 利用abort函数终止一个程序

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

printf("Calling abort()\n");

abort();

printf("Is the program be held?\n");

return 0;

}

例程说明：

本例程调用abort函数实现了一个程序的终止。实际上，该程序执行到abort语句处就被异常终止了，并没有执行后面两条语句。

本例程的运行结果为：

Calling abort()

Abnormal program termination

注意：

由于abort函数是异常终止一个进程，因此系统会将一条终止信息“Abnormal program termination”输出到stderr。

# atexit：注册终止函数

函数原型：int atexit(void \* func)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：注册终止函数。其中参数func为指向函数的指针。在程序正常终止时，系统会调用注册了的func函数。

返回值：注册成功返回0，否则返回非0。

例程如下： 利用函数atexit注册出口函数。

#include <stdio.h>

#include <stdlib.h>

void fun1(void)

{

printf("Exit function #1 called\n");

}

void fun2(void)

{

printf("Exit function #2 called\n");

}

int main(void)

{

atexit(fun1);

atexit(fun2);

return 0;

}

例程说明：

（1）首先，程序依次注册（登记）了两个出口函数fun1和fun2。

（2）当程序正常终止时，系统依次调用注册了的出口函数fun1和fun2。

本例程的运行结果为：

Exit function #2 called

Exit function #1 called

注意：

1、最多可以用atexit函数注册（登记）32个出口函数。

2、调用出口函数时，按照先注册后调用，后注册先调用的原则。因此本例程中先调用出口函数fun2，后调用出口函数fun1。

# bsearch：二分搜索函数

函数原型：void \*bsearch(const void \*key, const void \*base, size\_t \*nelem, size\_t width, int(\*fcmp)(const void \*, const \*))

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：二分法查找。参数key指向要查找的关键字的指针，base指向从小到大的次序存放元素的查找表，nelem指定查找表元素的个数，width指定查找表中每个元素的字节数，int(\*fcmp)(const void \*, const \*)为由用户提供的比较函数。

返回值：如果没有找到匹配的值返回0，否则返回匹配项的指针。

例程如下： 用二分法在有序数列中查找元素。

#include <stdio.h>

#include <stdlib.h>

int CMP(int \*a,int \*b)

{

if(\*a<\*b)

return -1;

else if(\*a>\*b)

return 1;

else

return 0;

}

int main(void)

{

int search[10]={1,3,6,7,10,11,13,19,28,56} ;

int a=13,\*p,i;

/\*对数组search进行二分搜索13\*/

p=(int \*)bsearch(&a, search,10, sizeof(int),CMP);

printf("The elems of the array are\n");

for(i=0;i<10;i++)

printf("%d ",search[i]);

/\*显示元素13在原数组中的位置\*/

if(p)

printf("\nThe elem 13 is located at %d of the array\n",p-search+1);

else

printf(“\nSearch is fail!!\n”);

getchar();

}

例程说明：

（1）首先，初始化一个查找数组search，在这里，该数组一定是按照键值从小到大排列的。

（2）利用函数bsearch进行二分法搜索。参数&a为要查找的关键字的指针，要查找的关键字a为13；参数search为查找表首地址；10为查找表元素个数；sizeof(int)为查找表中每个元素的字节数，大小为2个字节；CMP为比较函数的指针。

（3）将查找后的结果（匹配项的指针）赋值给指针变量p。

（4）显示元素13在原数组中的位置（由指针p和数组首地址search确定）。

本例程的运行结果为：

The elems of the array are

1 3 6 7 10 11 13 19 28 56

The elem 13 is located at 7 of the array

注意：

1、关于用户提供的比较函数：

bsearch函数需要用户提供一个比较搜索函数，该函数由bsearch函数调用，并向该比较搜索函数传递两个指针参数a和b。用户定义的比较函数必须在a<b时返回-1，在a=b时返回0，在a>b时返回1。这里的大于、小于、等于，完全由用户来定义。本例程中定义的大于、小于、等于就是数学上的大于、小于、等于。

2、二分法搜索简介：

二分法搜索又叫做折半搜索或折半查找。它是一种经典的顺序文件查找算法，要求查找表按关键字有序排列（从小到大或从大到小，bsearch函数要求从小到大排列）。其查找思想是：逐渐缩小查找范围，直至得到查找结果。查找过程为（以从小到大的序列为例）：将要查找的元素的关键字k与当前当前查找范围内位于居中的那个元素的关键字进行比较，若匹配，则查找成功，返回该元素的指针即可；否则，若查找元素的关键字k小于当前查找范围内位于居中的那个元素的关键字，则到当前查找范围的前半部分重复上述查找过程，若查找元素的关键字k大于当前查找范围内位于居中的那个元素的关键字，则到当前查找范围的后半部分重复上述查找过程。

二分法搜索的查找效率较顺序搜索的查找效率要高许多，因此在进行顺序文件的搜索查找中是很实用的。有关二分法搜索算法的详细讲述请参看数据结构、算法分析等书目。

# calloc：分配主存储器函数

函数原型：void \*calloc(size\_t nelem, size\_t size)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：分配并刷新内存。函数calloc有两个参数，第一个参数nelem指出要分配内存空间的项数；第二个参数size表明每一项的字节数。

返回值：分配成功返回第一个分配字节的指针，否则返回NULL。

例程如下： 利用函数calloc动态分配内存空间

#include<stdlib.h>

main()

{

int i,j,\*p=NULL;

printf("Please enter the size for allocation\n");

scanf("%d",&i);

p=(int \*)calloc(i,sizeof(int));

if(p)

{

printf("Please enter %d datas\n",i);

for(j=0;j<i;j++)

scanf("%d",&p[j]);

}

else {

printf("Allocation is fail\n");

return 0;

}

printf("The datas are\n");

for(j=0;j<i;j++)

printf("%d ",p[j]);

}

例程说明：

（1）本例程首先通过终端输入要开辟内存空间的大小，将其保留在变量i中。

（2）通过函数calloc动态分配内存空间（由用户指定i），并将分配空间的首地址赋值给指针变量p。

（3）如果分配成功（即p不为NULL），向该内存空间写入数据。

（4）打印出刚才写入的数据。

本例程的运行结果为：

Please enter the size for alloc

3

Please enter 3 datas

3 2 1

The datas are

3 2 1

注意：

本例程中分配内存空间时采用动态分配函数calloc，这样分配存储空间的方法同数组不同，可以由用户在程序中指定分配空间的大小，而不用像数组那样在程序编译前指定数组的大小。

# exit：正常终止进程函数

函数原型：void exit(int status)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：正常终止一个进程。参数status用来保存调用进程的出口状态，一般地，0表示正常退出，非0表示发生错误。

返回值：无返回值。

例程如下： 应用exit函数正常终止一个程序。

#include <stdlib.h>

#include <conio.h>

#include <stdio.h>

int main(void)

{

float a,b;

printf("Enter a\n");

scanf("%f",&a);

printf("Enter b\n");

scanf("%f",&b);

if(b)

printf("a/b=%f",a/b);

else

{

printf("Error:Divide by 0") ;

exit(0);

}

getchar();

return 1;

}

例程说明：

（1）首先程序提示输入两个数a和b，然后进行除法运算a/b。

（2）如果b不为0表明运算合法，输入a/b的结果。

（3）如果b为0，则运算不合法，于是在终端提示错误信息"Error:Divide by 0"，并应用exit函数终止该程序。

本例程的运行结果为：

Enter a

3

Enter b

0

Error:Divide by 0

# free：释放内存函数

函数原型：void free(void \*ptr)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：释放已分配的块。参数ptr为指向要释放的内存块的指针。

返回值：无返回值。

例程如下： 利用函数free释放内存空间。

#include<stdlib.h>

main()

{

char \*p;

p=(char \*)malloc(10\*sizeof(char));

strcpy(p,"Hello world\n");

printf("%s",p);

free(p);

p=NULL;

printf("%s",p);

}

例程说明：

（1）首先应用malloc函数在内存中分配一个10个字节大小的内存空间。

（2）将字符串"Hello world\n"复制到该内存空间中，并打印该字符串。

（3）释放掉p所指向的内存空间，将NULL赋值给p。

（4）显示p所指向的内容。

本例程的运行结果为：

Hello world

(null)

注意：

在使用完利用malloc或calloc函数分配的内存单元后，应该养成用free函数释放掉分配的内存单元的习惯。特别是在开发一些大型系统时，这样会很节省资源。

# getenv：获取环境变量

函数原型：char \*getenv(char \*envvar)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：获得环境字符串的首地址。

返回值：当envva所表示的环境变量存在时，返回其首地址；否则返回NULL。

例程如下： 显示环境变量

#include <stdlib.h>

#include <stdio.h>

int main(void)

{

char \*s;

s=getenv("COMSPEC");

printf("Command processor: %s\n",s);

return 0;

}

例程说明：

（1）首先利用函数getenv获取名为COMSPEC的环境字符串的首地址。

（2）根据得到的环境字符串的首地址，显示该环境字符串。

本例程的运行结果为：

Command processor: C:\WINDOWS\SYSTEM32\COMMAND.COM

注意：

C:\WINDOWS\SYSTEM32\COMMAND.COM即为要得到的环境字符串。

# malloc：动态分配内存函数

函数原型：void \*malloc(unsigned size)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：动态分配一块内存空间，size为指定的分配空间的大小（字节数）。

返回值：分配成功，则返回指向分配内存的指针，否则返回NULL。

例程如下： 利用函数malloc动态分配内存空间

#include<stdlib.h>

main()

{

char \*str;

if ((str = malloc(15)) == NULL)

{

printf("Not enough memory to allocate buffer\n");

exit(1);

}

strcpy(str, "Hello World!");

printf("String is %s\n", str);

free(str);

return 0;

}

例程说明：

（1）本例程首先利用函数malloc分配一个15个字节大小的内存空间，并将其首地址赋值给指针型变量str。

（2）如果分配成功，复制字符串"Hello World!"到刚刚分配好的内存缓冲区中。

（3）在屏幕上打印该字符串。

本例程的运行结果为：

String is Hello World!

# qsort：快速排序函数

函数原型：void qsort(void \*base, int nelem, int width, int(\*fcmp)(const void \*, const \*))

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：对记录进行从小到大的快速排序。参数base指向存放待排序列的数组的首地址，nelem为数组中元素的个数，width为每个元素的字节数，int(\*fcmp)(const void \*, const \*)为由用户提供的比较函数。

返回值：无

例程如下：利用qsort函数对无序序列进行快速排序（从小到大排序）。

#include <stdio.h>

#include <stdlib.h>

int CMP(int \*a,int \*b)

{

if(\*a<\*b)

return -1;

else if(\*a>\*b)

return 1;

else

return 0;

}

int main(void)

{

int sort[10]={3,2,6,12,1,7,-5,9,30,16} ;

int i;

printf("\nThe array that is before sort\n");

for(i=0;i<10;i++)

printf("%d ",sort[i]);

qsort(sort,10,sizeof(int),CMP) ;

printf("\nThe array that is after sort\n");

for(i=0;i<10;i++)

printf("%d ",sort[i]);

getchar();

}

例程说明：

（1）首先初始化待排序列sort。

（2）显示最初数组sort中的元素排列情况。

（3）对数列sort进行快速排序，这里sort为数组sort的首地址；10为数组sort的元素个数；sizeof(int) 为待排序列中每个元素的字节数，大小为2个字节；CMP为用户定义的比较函数的指针。

（4）最后显示出排序后（从小到大排序）数组sort中的元素排列情况。

本例程的运行结果为：

The array that is before sort

3 2 6 12 1 7 -5 9 30 16

The array that is after sort

-5 1 2 3 6 7 9 12 16 30

注意：

1、关于用户提供的比较函数：

qsort函数需要用户提供一个比较搜索函数。该函数由qsort函数调用，并向该比较搜索函数传递两个指针参数a和b。用户定义的比较函数必须在a<b时返回-1，在a=b时返回0，在a>b时返回1。这里的大于、小于、等于，完全由用户来定义。本例程中定义的大于、小于、等于就是数学上的大于、小于、等于。

2、快速排序简介：

快速排序是一种经典的高效排序算法。它是取待排序列中某个元素为基准，按照该元素值的大小将整个序列划分为左右两个子序列，其中左子序列的值小于或等于基准元素的值；右子序列的值大于或等于基准元素的值。然后分别对两个子序列重复上述排序过程，直至所有元素都排在相应位置为止。

有关快速排序算法的详细介绍，请参看数据结构、算法分析等书目。

# realloc：重新分配主存函数

函数原型：void \*realloc(void \*ptr, unsigned newsize)

头文件：#include<stdlib.h>

是否是标准函数：是

函数功能：重新分配内存空间。第一个参数ptr为一个指针，它指向重新设定大小的块；第二个参数newsize为重新分配内存的字节大小。

返回值：分配成功，则返回修改块的指针，否则返回NULL。

例程如下： 利用函数realloc重新分配内存空间

#include<stdlib.h>

main()

{

int \*sqlist,i,len;

len=10;

sqlist=(int \*)malloc(len\*sizeof(int));

for(i=0;i<20;i++)

{

if(i>=len){

len=len\*2;

sqlist=realloc(sqlist,len\*sizeof(int));

}

sqlist[i]=i;

}

for(i=0;i<20;i++)

printf("%d ",sqlist[i]);

}

例程说明：

（1）本例程首先分配一个只有10个整型数据大小的内存空间。

（2）然后通过程序向该数组输入20个整数。在这里要加一个判断，即当输入的数据超过原来分配的内存空间的长度时，调用realloc函数将内存重新分配，大小为上一次长度的2倍。这样第i次调用realloc函数时分配的内存长度为10\*2i个整型变量长度。

（3）最后打印这20个整数。

本例程的运行结果为：

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

注意：

1、realloc函数的作用实际上分为两步，一是在内存中重新开辟指定大小空间；二是将原内存空间的数据复制到新开辟的空间中（这是在新分配的内存比原内存大的情况下）。

2、如果新分配的内存比原内存小，则新分配的内存单元不被初始化。

3、realloc函数多用于动态顺序表这种数据结构的建立。