**vs\_32位和64位系统中c数据类型大小**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 大小 | 数据类型 | 大小 |
| int | 4B | char | 1B |
| unsigned int | 4B | bool | 不支持 |
| short=short int | 2B | long long | 8B |
| unsigned short=unsigned short int | 2B | float | 4B |
| long=long int | 4B | double | 8B |
| unsigned long= unsigned short int | 4B | long double | 8B |
| 指针 | 32位4B，64位8B | sizeof | 32位4B，64位8B |

**GCC\_64位的c数据类型及大小**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 大小 | 数据类型 | 大小 |
| int | 4B | char | 1B |
| unsigned int | 4B | bool | 1B |
| short=short int | 2B | long long | 8B |
| unsigned short=unsigned short int | 2B | float | 4B |
| long=long int | 8B | double | 8B |
| unsigned long= unsigned short int | 8B | long double | 16B |
| 指针 | 32位4B，64位8B | sizeof | 32位4B，64位8B |

**数组和指针**

1. 赋值运算符的优先级是最低的，赋值的左值必须时变量
2. 如if（j==5||i=3）相当于if（(j==5||i)=3）
3. 标准输入/输出函数

|  |
| --- |
| **#include <stdio.h>**  **int scanf( const char \*format, ... );** |

功能：scanf()函数根据由format(格式)指定的格式从stdin(标准输入)读取,并保存数据到其它参数. 它和[printf()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdio_details.html#printf)有点类似.format(格式)字符串由控制字符,空白字符和非空白字符组成. scanf()读取匹配format(格式)字符串的输入.当读取到一个控制字符,它把值放置到下一个变量.空白(tabs, 空格等等)会跳过.非空白字符和输入匹配,然后丢弃.如果是一个在%符号和控制符间的数量,那么只有指定数量的字符转换到变量中. 如果scanf()遇到一个字符集(用%[]控制字符表示), 那么在括号中的任意字符都会读取到变量中.scanf()的返回值是成功赋值的变量数量, 发生错误时返回EOF. 控制字符以一个%符号开始,如下:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| %d | 一个有符号十进制int整数 |  | %c | 一个单一的字符 |
| %i | 一个有符号int整数 |  | %p | 一个指针 |
| %u | 一个无符号int整数 |  | %n | 一个等于读取字符数量的整数 |
| %o | 一个无符号int八进制数 |  | %% | 一个精度符号'%' |
| %x | 一个无符号int十六进制数 |  | %[] | 一个字符集 |
| %f，%e，%g | 一个浮点数，可以是小数，指数 | 默认5位 |  |  |
| %s | 一个字符串 | 遇到空格、回车、TAB就结束读取，自动添加\0 | | |

修饰符，

l表示长型;

h表示短型，

\*在scanf中表示读取后不赋给变量，在printf中表示域宽由变量决定,如

printf(“%\*s%s”，width，””,hello”)可在hello前输出width个空格

m域宽，

n表示小数位数或者截取字串的长度，

-表示左对齐。

注意：scanf输入时必须严格按照格式串中输入。不能加减空格。

|  |
| --- |
| **#include <stdio.h>**  **int printf( const char \*format, ... );** |

功能：printf()函数根据format(格式)给出的格式打印输出到STDOUT(标准输出)和其它参数中.

字符串format(格式)由两类项目组成 - 显示到屏幕上的字符和定义printf()显示的其它参数. 基本上, 你可以指定一个包含文本在内的format(格式)字符串,也可以是映射到printf()其它参数的"特殊"字符. 例如本代码

char name[20] = "Bob";

int age = 21;

printf( "Hello %s, you are %d years old\n", name, age );

显示下列输出:

Hello Bob, you are 21 years old

%s 表示, "在这里插入首个参数,一个字符串." %d 表示第二个参数(一个整数)应该放置在那里. 不同的"%-codes"表示不同的变量类型, 也可以限制变量的长度.

|  |  |  |  |
| --- | --- | --- | --- |
| %d | 带符号int整数 | %c | 字符 |
| %i | 带符号int整数 | %s | 一串字符，遇到\0就停止输出 |
| %u | 无符号int整数 | %o | 八进制int整数 |
| %f | 浮点数 | %x | 无符号十六进制int整数, 用小写字母 |
| %e | 科学计数法, 使用小写"e" | %X | 无符号十六进制int整数, 用大写字母 |
| %E | 科学计数法, 使用大写"E" | %p | 一个指针 |
| %g | 使用%e或%f中较短的一个 | %n | 参数是一个指向一个整数的指针.指向的是字符数放置的位置 |
| %G | 使用%E或%f中较短的一个 | %% | 一个精度符号'%' |

一个位于一个%和格式化命令间的整数担当着一个最小字段宽度说明符,并且加上足够多的空格或0使输出足够长. 如果你想填充0,在最小字段宽度说明符前放置0. 你可以使用一个精度修饰符,它可以根据使用的格式代码而有不同的含义.

用%e, %E和 %f,精度修饰符让你指定想要的小数位数. 例如,

%12.6f

将会至少显示12位数字,并带有6位小数的浮点数.

用%g和 %G, 精度修饰符决定显示的有效数的位数最大值.

用%s,精度修饰符简单的表示一个最大的最大长度, 以补充句点前的最小字段长度.

所有的printf()的输出都是右对齐的,除非你在%符号后放置了负号. 例如,

%-12.4f

将会显示12位字符,4位小数位的浮点数并且左对齐. 你可以修改带字母l和h 的%d, %i, %o, %u和 %x 等类型说明符指定长型和短型数据类型 (例如 %hd 表示一个短整数). %e, %f和 %g 类型说明符,可以在它们前面放置l指出跟随的是一个double. %g, %f和 %e 类型说明符可以置于字符'#'前保证出现小数点, 即使没有小数位. 带%x类型说明符的'#'字符的使用, 表示显示十六进制数时应该带'0x'前缀. 带%o类型说明符的'#'字符的使用, 表示显示八进制数时应该带一个'0'前缀.

你可以在输出字符串中包含 [连续的Escape序列](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/escape_sequences.html).

printf()的返回值是打印的字符数,如果发生错误则返回一个负值.

|  |
| --- |
| **#include <stdio.h>**  **int getchar( void );** |

功能：getchar()函数从STDIN(标准输入)获取并返回下一个字符,如果到达文件尾返回EOF.

|  |
| --- |
| **#include <stdio.h>**  **int putchar( int ch );** |

功能：putchar()函数把ch写到STDOUT(标准输出). 代码 putchar( ch )和 putc( ch, STDOUT )一样. putchar()的返回值是被写的字符, 发生错误时返回EOF.

1. scanf 读取数值时是自动忽略\n，而读取字符时不会忽略。%c,每次读取一个字符，%s一次读取字符串，遇到\n，停止读取。Scanf错误时返回EOF，gets 出错时返回NULL。Scanf遇到空格，回车TAB，都会停止读取，而gets则不会。

|  |  |  |
| --- | --- | --- |
| 转义字符 | 意义 | ASCII码值（十进制） |
| \a | 响铃(BEL) | 007 |
| \b | 退格(BS) ，将当前位置移到前一列 | 008 |
| \f | 换页(FF)，将当前位置移到下页开头 | 012 |
| \n | 换行(LF) ，将当前位置移到下一行开头 | 010 |
| \r | 回车(CR) ，将当前位置移到本行开头 | 013 |
| \t | 水平制表(HT) （跳到下一个TAB位置） | 009 |
| \v | 垂直制表(VT) | 011 |
| \\ | 代表一个反斜线字符''\' | 092 |
| \' | 代表一个单引号（撇号）字符 | 039 |
| \" | 代表一个双引号字符 | 034 |
| \? | 代表一个问号 | 063 |
| \0 | 空字符(NULL) | 000 |
| \ooo | 1到3位八进制数所代表的任意字符 | 三位八进制 |
| \xhh | 1到2位十六进制所代表的任意字符 | 二位十六进制 |

1. 正数原码=反码=补码，负数反码是将符号位以外求反，负数补码是对其反码加1，如出现进位应进位到符号位为止。
2. 补码的作用一是为了运行减法[A-B]补码=A补+（-B）补码，二是统一+0和-0。数字在计算机中是用补码表示的。
3. 一维数组下标必须为常量，或者定义同时赋初值则可以省略下标。数值型数组不赋值默认为0，字符型数值默认为\n。可以一次赋值一个，也可以整体赋值。a[]={1，2，3，4，,5}
4. rand()产生0-最大范围的随机数。a+rand()%(b-a+1)产生a到b的随机整数数；若在调用rand()之前调用 srand（time(NULL)），则每次生成的随机数都不一样。调用time函数应当包含time.h头文件。
5. 一维数组传递的时候会退化为指针。所以一维数组传递时必须同时传入长度；一维数组作为形参，应该空下标；而二维数组传递时会退化为数组指针，因而必须传入数组的第一个下标或者两个下标；而二位数组作形参应该空一维下标
6. 左移运算：算术和逻辑左移一样，右边补0；右移运算>>：算术右移要考虑符号位，逻辑右移直接补0
7. scanf输入字符串时如果输入的字符数小于数组大小就会越界，非法占用其余内存空间，scanf会自动补\O，并且scanf遇到空格回车制表符就结束读取。
8. 指针的使用场景就是偏移和传递，否则不需要使用指针。
9. 指针存放的是对象的地址。对象可以是函数，数组，数组元素，变量，结构体变量，共用体变量，枚举变量等。。。
10. Int \*p;int a[10];p=a;则p[i]等价a[i]等价\*（p+i）等价\*（a+i）;
11. Char a[]=”I love you”；int \*p=a；与char \*p=“I love you”有本质的差别，前者是将字符串赋值给数组，此时P[i]是可以修改的，而后者是将常量的地址赋值给指针p，而此时P[i]是不可以修改的。即指向常量的指针和指向变量的指针。
12. 动态内存申请和释放 ：

（数据类型 \*）malloc（大小），free(指针)

（数据类型\*）realloc（原始指针，新的大小）,成返回地址，失败返回NULL；

（数据类型\*）calloc（空间个数，每个空间大小）申请空间个数\*每个空间大小这么多空间并返回地址

memset（空间起始地址，要赋予的值，空间大小）将一段空间赋予值

memcpy（目标地址，原地址，大小）拷贝内存空间

memmove（目标地址，原地址，大小）移动内存空间

注意free（指针）指针必须是申请的指针如：char \*p=(char \*)malloc(10);p++；free（p）则会出错。另外调用free（p），p所指的内存空间虽然释放了，但值不会改变。仍可以被使用，这时p就是野指针。因而返回值为指针时，该指针不能指向栈空间应该指向堆空间。

1. memset(指针，赋值，空间大小)对指针所指向的空间大小得每个内存单元赋初始值。
2. 几个概念的区别：

（1）指针数组和数组指针：

指针数组是数组元素为指针的数组，数据类型 \*数组名[size]，主要用于对指针所指内容进行排序而不移动元素。如：char \*a[10]

数组指针是指向数组的指针，数据类型 （\*指针名）[size]，指针+1，偏移整个数组的大小.如int a[3][3],int（\*p）[3],p=a;

（2）指针函数与函数指针

指针函数 存储类型 返回值类型 \*函数名（形参表）；

函数指针 存储类型 返回值类型 （\*指针名）（形参表）；

函数指针的作用在于选择不同的函数来执行。如对两个数进行加减乘除运算，就可以定义一个指针分别指向这四个函数。或者使用函数指针作为函数参数，以传递不同的行为。

1. 对于实参是空指针，形参也为指针，则函数调用时候会出错，因而c中只能用二级指针在子函数中来对主函数的实参指针进行修改。否则就不需要使用二级指针。

如：

void fun（char \*p）

{

int i;

p=&i;

}

void main()

{

char \*p;

fun(p);

}

这里报错原因是因为传递的是p的值，而p的值是空的。

则调用函数会出错。正确的用法应该是用二级指针，这样尽管p的值是空的，但由于传递的是p所在地址因而不会报错。

Void fun（char \*\*p）

{ int i;

\*pp=&i

}

void main()

{

char \*p；

fun（&p）；

}

20 fflush(stdin)清除缓冲区，fflush(stdout)将缓冲区的数据输出后清除缓冲区。返回值为int。

1. 指针排序

对于链表来说，由于移动元素复杂， 因而可以对指向各节点的指针进行排序。

22 指针（变量）

指向某内存空间的变量，指针的值是所指对象的地址，二级指针的值就是一级指针的地址

**函数与自定义数据类型**

**1.**变量的作用域和生存期

存储类型 数据类型 变量名

函数或语句块内部：auto 局部作用域，动态生存期，存放在动态存储区；(默认)

Static 局部作用域，静态生存期，存放在静态存储区；

register 局部作用域，动态生存期，放在cpu的寄存器中，register变量不能取址。

函数外部：static 文件作用域，静态生存期，存放在静态存储区；

extern 全局作用域，静态生存期，存放在静态存储区；（默认，实质是变量的声明而不是定义，因为不需申请空间）

**2.**函数的作用域和生存期

存储类型 返回值类型 函数名（形参表）

static 文件作用域，静态生存期；

extern 全局作用域，静态生存期；（默认，可以省略）

**3.**可以将不同的函数模块放在不同的源文件中，但每个源文件都是分别编译，因而必须分别添加头文件。分别编译可以提高编译效率，在对某个源文件修改编译的时候只需单独修改编译某一个源文件。

**4.**c中的参数传递

传值：默认的传值，即在函数被调用的时候，给形参申请一个空间，再将实参的值传递给形参，对形参的任何改变不会影响实参数的值；无论传址、传址、传输组都是传实参的内容（变量的值或指针的值）而不会传地址。传址只是人为的传递将地址的值传递。

传址：所谓传址又叫做传指针，即在函数被调用的时候，给形参指针开辟一个空间用来存放传递过来的地址，将实参所在的内存地址传递给形参，对形参的任何改变也不会影响实参所指的内容，但是对形参所指的内容的改变将会影响到实参所指的内容（因为这两个指针都指向同一个内存空间）

传数组：实质是传址，将数组的起始地址传给形参，会退化为指针丢失数组长度，对形参的改变会改变实参。

**5.** 一个程序的运行起来后，其在内存中有5个区域  
 ①程序代码区：这个很简单，代码要执行，肯定要加载进内存，我们不必关心。

②文字常量区：一般我们这样定义一个字符串时，其是在文字常量区的：

char\*s1="hello,world";

char\*s2="hello,world";  
if(s1==s2)  
printf("s1和s2指向同一个在文字常量区的字符串");

这里，s1和s2指向的是同一个字符串，即"hello,world"在内存中只有一份。

③静态存储区：静态生存期的变量会放在这个区域。

以上三个区域的内存在程序的时候就开辟好了的。

④栈：局部变量就是在栈里的。另外，函数调用时的参数也是在栈里的，这个现在不必关心  
⑤堆：malloc或new出来的内存就是在堆里的，需要程序员自己管理清除。或程序结束时由os回收。

**5.**①尽量不要把函数放在循环中

②尽量不要在子函数中定义数据结构，而通过指针传给子函数

③能用栈区就不用堆区，因为malloc申请堆区的时间开销远大于在栈区定义一个变量的的时间开销。

**6.**

|  |
| --- |
| **#include <setjmp.h>**  **int setjmp( jmp\_buf envbuf );** |

**功能：** 函数将系统栈保存于*envbuf*中，以供以后调用[longjmp()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdother_details.html#longjmp)。返回值是记录保存系统栈的次数。当第一次调用setjmp(),它的返回值为零。之后调用longjmp(),longjmp()的第二个参数即为setjmp()的返回值。

|  |
| --- |
| **#include <setjmp.h>**  **void longjmp( jmp\_buf envbuf, int status );** |

**功能：** 函数使程序从前次对[setjmp()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdother_details.html#setjmp)的调用处继续执行。参数*envbuf*一般通过调用[setjmp()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdother_details.html#setjmp)设定。参数*status* 为[setjmp()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdother_details.html#setjmp)的返回值，用来指示不同地点longjmp()的执行. *status* 不能设定为零。

使用setjmp和longjmp要注意以下几点：

（1）setjmp与longjmp结合使用时，它们必须有严格的先后执行顺序，也即先调用setjmp函数，之后再调用longjmp函数，以恢复到先前被保存的“程序执行点”。否则，如果在setjmp调用之前，执行longjmp函数，将导致程序的执行流变的不可预测，很容易导致程序崩溃而退出

（2） longjmp必须在setjmp调用之后，而且longjmp必须在setjmp的作用域之内。具体来说，在一个函数中使用setjmp来初始化一个全局标号，然后只要该函数未曾返回（结束），那么在其它任何地方都可以通过longjmp调用来跳转到 setjmp的下一条语句执行。实际上setjmp函数将发生调用处的局部环境保存在了一个jmp\_buf的结构当中，只要主调函数中对应的内存未曾释放 （函数返回时局部内存就失效了），那么在调用longjmp的时候就可以根据已保存的jmp\_buf参数恢复到setjmp的地方执行。

**7.**函数不能反悔动态生存期的对象。因为动态生存期的对象在函数调用结束后会即刻销毁。

**8.**结构体共用体枚举

结构体：每个成员独享一个空间，大小为所有成员大小之和；struct。

共用体：所用成员共享一个空间，大小为最大成员的大小；所有成员的地址都相同，共用体的值等于最新赋值的成员，union。

枚举：枚举中每个成员代表一个常量，不可对成员赋值，默认从0开始依次递增，或者从指定值依次递增。

enum。

**9.**野指针 ：指向不属于自己的内存空间的指针。

**10.**

|  |
| --- |
| **#include <stdlib.h>**  **void qsort( void \*buf, size\_t num, size\_t size, int (\*compare)(const void \*, const void \*) );** |

**功能：** 对buf 指向的数据(包含num 项num从1开始计数,每项的大小为size)进行快速排序。如果函数compare 的第一个参数小于第二个参数，返回负值；如果等于返回零值；如果大于返回正值。函数对buf 指向的数据按升序排序。如对int a[100]排序：

int comp(const void\*a,const void\*b)

return \*(int\*)a-\*(int\*)b;

注意：qsort传给cmp函数的参数是指向数组元素的指针。const void\*a,const void\*b可自己根据需要改变类型。因而数组元素是变量，定义为 1级指针，数组元素是一级指针，则定义为2级指针，数组元素是二级指针，强定义为三级指针。

**11.**可以自己写头文件，头文件以“.h”为扩展名，若放在库中用#include<文件名>来引用，若放在工程下以#include”文件名”来引用。头文件的作用就是编译的时候将头文件的内容来覆盖#include的位置。头文件中可以定义变量,函数，等，但不推荐。因为程序编译是以源文件为单位的，只有当头文件被包含在某个源文件内才参与编译。一个源文件可能包含多个头文件，这些头文件可能都包含了某个头文件“\*.h”，那么这个头文件必须用预处理语句#ifndef, #define, #endif 防止被重复包含，否则这个源文件内就会包含多个该头文件“\*.h”，假设该头文件包含一个变量或者函数定义，那么在该源文件内就存在多个同样内容的定义，这样就会造成重复定义。因而在平常时候，我们不要将函数定义在头文件中。只在头文件中进行声明，在源文件中实现。这样对于大型工程，有助于代码文件的有效管理。所以.h文件中一般只能包含全局变量的声明，函数声明，宏定义一类的，在.h文件中定义变量是不被推荐的。

下面给一个#ifndef/#define/#endif的格式：

#ifndef A\_H //意思是"if not define a.h" 如果不存在a.h

#define A\_H //就引入a.h

#include <math.h> // 引用标准库的头文件

#include “header.h” // 引用非标准库的头文件

void Function1(…); // 全局函数声明

#endif //否则不需要引入

**12.**

|  |
| --- |
| **#include <time.h>**  **time\_t time( time\_t \*time );** |

功能： 函数返回当前时间，如果发生错误返回零。如果给定参数time ，那么当前时间存储到参数time中。返回的时距离1970年0时0分0分的时间间隔。

|  |
| --- |
| **#include <time.h>**  **clock\_t ( void );** |

功能：函数返回自程序开始运行的处理器时间，如果无可用信息，返回-1。 转换返回值以秒记, 返回值除以CLOCKS\_PER\_SECOND. (注: 如果编译器是POSIX兼容的, CLOCKS\_PER\_SECOND定义为1000000.) 返回值应该是cpu的时钟。

**文件（stdio.h）**

1. 文件：程序和数据的集合就叫文件。
2. 文件缓冲区：由于内存和I/O设备的的速度差异很大。因而在内存中设置输入、输出缓冲区。读取时，先从输入设备中读取一批数据到输入缓冲区，然后在依次送给变量。输出时先存放在输出缓冲区，待缓冲区满或程序结束或主动刷新时才输出到输出设备。
3. 文件信息区：

每个打开一个文件，都会在内存中开辟一个文件信息区用于存放文件的相关相关属性。这些信息存放在一个名为FILE的结构体中。

1. 标准I/O流与文件流

标准输入流、标准输出流、标出出错输出流（键盘，屏幕，屏幕）如putchar，getchar，putc，getc，puts，gets，printf，scanf。

文件流（文本文件与二进制文件）如fputc，fgetc，fputs，fgets，fprintf，fscanf。

实质就是流指针的不同，前者流指针为stdin或stdout，后者则为FILE型指针。

1. 文件的存储形式
2. ASCII文件，又叫文本文件，内存数据（二进制）——>ASCII码——>硬盘
3. 二进制文件，又叫映像文件，内存数据（二进制）——>硬盘
4. 打开/关闭文件

|  |
| --- |
| **FILE \*fopen( const char \*fname, const char \*mode );** |

功能：fopen()函数打开由fname(文件名)指定的文件, 并返回一个关联该文件的流.如果发生错误, fopen()返回NULL. mode(方式)是用于决定文件的用途(例如 用于输入,输出,等等) ，文件名包括路径、主文件名、扩展名。

|  |  |  |
| --- | --- | --- |
| "r" | 以只读方式打开文本文件 | 该文件必须存在，否则打开时出错。 |
| "w" | 以只写方式打开文本文件 | 打开时若文件存在则清空该文件，若文件不存在则建立该文件。 |
| "a" | 以附加的方式打开只写文件 | 打开时若文件不存在则建立该文件，如果存在则写入的数据会被加到文件尾。（原EOF保留） |
| "rb" | 以只读方式打开二进制文件 | 该文件必须存在，否则打开时出错。 |
| "wb" | 以只读方式打开二进制文件 | 打开时若文件存在则清空该文件，若文件不存在则建立该文件。 |
| "ab" | 以只读方式打开二进制文件 | 打开时若文件不存在则建立该文件，如果存在则写入的数据会被加到文件尾。（原EOF保留） |
| "r+" | 以可读写方式打开文本文件 | 该文件必须存在，否则打开时出错。 |
| "w+" | 以可读写方式打开文本文件 | 打开时若文件存在则清空该文件，若文件不存在则建立该文件。 |
| "a+" | 以附加的方式打开可读写文本文件 | 打开时若文件不存在则建立该文件，如果存在则写入的数据会被加到文件尾。（原EOF不保留） |
| "rb+" | 以可读写方式打开二进制文件 | 该文件必须存在，否则打开时出错。 |
| "wb+" | 以可读写方式打开二进制文件 | 打开时若文件存在则清空该文件，若文件不存在则建立该文件。 |
| "ab+" | 以附加方式打开可读写二进制文件 | 打开时若文件不存在则建立该文件，如果存在则写入的数据会被加到文件尾。（原EOF不保留） |

|  |
| --- |
| **int fclose( FILE \*stream );** |

功能：函数fclose()关闭给出的文件流, 释放已关联到流的所有缓冲区. fclose()执行成功时返回0,否则返回EOF.

FILE \*fp；//定义一个文件指针

if（!(fp=fopen（”文件名”，打开方式）)）

printf(”打开出出错”)

…………

fclose（fp）//

1. 顺序读/写文件

（1）对文本文件的顺序读写

|  |
| --- |
| **int fgetc( FILE \*stream );** |

功能：fgetc()函数返回来自stream(流)中的下一个字符,如果到达文件尾或者发生错误时返回EOF.

注意： fgetc与getc一样自动将文本文件的\r\n转为\n读入，将文本文件的\0读入，将标准输入的\n读入。

|  |
| --- |
| **int fputc( int ch, FILE \*stream );** |

功能：函数fputc()把给出的字符ch写到给出的输出流. 返回值是字符, 发生错误时返回值是EOF.

注意： fputc会把\0输出到文本文件和屏幕；将\n输出到文本文件作为\r\n，输出到屏幕作为\n。

|  |
| --- |
| **char \*fgets( char \*str, int num, FILE \*stream );** |

功能：函数fgets()从给出的文件流中读取[num - 1]个字符并且把它们转储到str(字符串)中。若当前行字符数大于[num - 1]则下次接着读，自动将str[num]置为\0；若当前行字符少于[num - 1]，fgets()在到达行末时停止,在这种情况下,str(字符串)将会被一个换行符\n结束并且自动将str[num]置为\0。 如果fgets()遇到EOF, str(字符串)将会以\0结束.fgets()成功时返回str(字符串),失败时返回NULL.

注意： gets()读入时将舍去换行符\n，仅用于标准输入，仅有一个参数； fgets()可用于标准输入和文本文件读入，自动将文本文件的\r\n和标准输入的\n转为\n，且读入换行符\n，有3个参数。

|  |
| --- |
| **int fputs( const char \*str, FILE \*stream );** |

功能：fputs()函数把str(字符串)指向的字符写到给出的输出流. 成功时返回非负值, 失败时返回EOF.

注意：fputs输出\n到文本文件作为\r\n，到屏幕作为\n，遇到\0就停止输出且不会输出\0，有两个参数。fputs向文件和屏幕输出时不会自动换行。

puts仅用于标准输出，遇到\0就停止输出且不会输出\0。puts向屏幕输出时会自动换行。

|  |
| --- |
| **int fscanf( FILE \*stream, const char \*format, ... );** |

功能：函数fscanf()以[scanf()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdio_details.html#scanf)的执行方式从给出的文件流中读取数据. fscanf()的返回值是事实上已赋值的变量的数,如果未进行任何分配时返回EOF.

|  |
| --- |
| **int fprintf( FILE \*stream, const char \*format, ... );** |

功能：fprintf()函数根据指定的format(格式)(格式)发送信息(参数)到由stream(流)指定的文件. fprintf()只能和[printf()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdio_details.html#printf)一样工作. fprintf()的返回值是输出的字符数,发生错误时返回一个负值.

|  |
| --- |
| **int feof( FILE \*stream );** |

功能：函数feof()在到达给出的文件流的文件尾时返回一个非零值. 否则返回零。

（2）对二进制文件的顺序读写

|  |
| --- |
| **int fread( void \*buffer, size\_t size, size\_t num, FILE \*stream );** |

功能：函数fread()读取[num]个对象(每个对象大小为size(大小)指定的字节数),并把它们替换到由buffer(缓冲区)指定的数组. 数据来自给出的输入流. 函数的返回值是读取的内容数量.

使用[feof()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdio_details.html#feof)或[ferror()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdio_details.html#ferror)判断到底发生哪个错误.

|  |
| --- |
| **int fwrite( const void \*buffer, size\_t size, size\_t count, FILE \*stream );** |

功能：fwrite()函数从数组buffer(缓冲区)中, 写count个大小为size(大小)的对象到stream(流)指定的流. 返回值是已写的对象的数量.

1. 随机读写文件

随机读写文件的实质就是改变文件标记位置的顺序读取

|  |
| --- |
| **void rewind( FILE \*stream );** |

功能：函数rewind()把文件指针移到由stream(流)指定的开始处, 同时清除和流相关的错误和EOF标记.

|  |
| --- |
| **int fsetpos( FILE \*stream, const fpos\_t \*position );** |

功能：fsetpos()函数把给出的流的位置指针移到由position对象指定的位置. fpos\_t是在stdio.h中定义的. fsetpos()执行成功返回0,失败时返回非零.

|  |
| --- |
| **int fgetpos( FILE \*stream, fpos\_t \*position );** |

功能：fgetpos()函数保存给出的文件流(stream)的位置指针到给出的位置变量(position)中. position变量是fpos\_t类型的(它在stdio.h中定义)并且是可以控制在FILE中每个可能的位置对象. fgetpos()执行成功时返回0,失败时返回一个非零值.

|  |
| --- |
| **int fseek( FILE \*stream, long offset, int origin );** |

功能：函数fseek()为给出的流设置位置数据. origin的值应该是下列值其中之一(在stdio.h中定义):

|  |  |  |
| --- | --- | --- |
| 0 | SEEK\_SET | 从文件的开始处开始搜索 |
| 1 | SEEK\_CUR | 从当前位置开始搜索 |
| 2 | SEEK\_END | 从文件的结束处开始搜索 |

fseek()成功时返回0,失败时返回非零. 你可以使用fseek()移动超过一个文件,但是不能在开始处之前. 使用fseek()清除关联到流的EOF标记. offset必须是长整型。

注意：如果连续读或连续写不需用fseek刷新文件标记位置，但如果读后写或者写后读就必须刷新文件标记位置。

|  |
| --- |
| **long ftell( FILE \*stream );** |

功能：ftell()函数返回stream(流)当前的文件位置,如果发生错误返回-1.

1. 文件读/写出错

|  |
| --- |
| **int ferror( FILE \*stream );** |

功能：ferror()函数检查stream(流)中的错误, 如果没发生错误返回0,否则返回非零. 如果发生错误, 使用[perror()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdio_details.html#perror)检测发生什么错误.

|  |
| --- |
| **void perror( const char \*str );** |

功能：perror()函数打印str(字符串)和一个相应的执行定义的错误消息到全局变量errno中.

|  |
| --- |
| **void clearerr( FILE \*stream );** |

功能：clearerr函数重置错误标记和给出的流的EOF指针. 当发生错误时,你可以使用[perror()](mk:@MSITStore:C:\Users\LIAOWEIZHI\Desktop\C／C++函数大全.chm::/cppreference.com/stdio_details.html#perror)判断实际上发生了何种错误

1. 刷新输入/输出缓冲区

|  |
| --- |
| **int fflush( FILE \*stream );** |

功能：如果给出的文件流是一个输出流,那么fflush()把输出到缓冲区的内容写入文件. 如果给出的文件流是输入类型的,那么fflush(STDIN)会清除输入缓冲区. fflush()在调试时很实用,特别是对于在程序中输出到屏幕前发生错误片段时. 直接调用 fflush( STDOUT )输出可以保证你的调试输出可以在正确的时间输出.

11. 二进制和文本模式的区别

　　1.在windows系统中，文本文件下，文件以"\r\n"代表换行。若以文本模式打开文件，并用fputs等函数写入换行符"\n"时，函数会自动在"\n"前面加上"\r"。即实际写入文件的是"\r\n" 。

2.在类Unix/Linux系统中文本模式下，文件以"\n"代表换行。所以Linux系统中在文本模式和二进制模式下并无区别。

12. main(int argc,char \*argv[])命令行参数在属性-》调试-》命令行参数中设置

argc：由系统传给函数的程序清单数。若argc==3则：

argv[0] 指向程序运行的全路径名

argv[1] 指向在DOS命令行中执行程序名后的第一个字符串

argv[2] 指向执行程序名后的第二个字符串

13.

|  |
| --- |
| **int remove( const char \*fname );** |

remove()函数删除由fname(文件名)指定的文件. remove()成功时返回0,如果发生错误返回非零.