想了解全部内容请移步至

<http://www.cplusplus.com/>

<http://zh.cppreference.com/w/%E9%A6%96%E9%A1%B5>

<https://www.tutorialspoint.com/c_standard_library/>

<https://msdn.microsoft.com/en-us/library/hh875057.aspx>

<http://pubs.opengroup.org/onlinepubs/9699919799/>

<https://www-s.acm.illinois.edu/webmonkeys/book/c_guide/>

|  |  |
| --- | --- |
| B | <https://www.bell-labs.com/usr/dmr/www/kbman.html> |
| c89(withdrawn) | <https://share.ansi.org/Shared%20Documents/Standards%20Action/2004%20PDFs/SAV3528.pdf> |
| c90(withdrawn) | <http://www.iso.org/iso/catalogue_detail.htm?csnumber=17782> |
| c99(withdrawn) | <http://www.iso.org/iso/catalogue_detail.htm?csnumber=29237> |
| c11 | <http://www.iso.org/iso/catalogue_detail.htm?csnumber=57853>  <http://download.csdn.net/detail/sfgassdfg/9570151#comment> |

更新地址：<https://github.com/jhcarl0814/notes>

若发现文档内容与实际不符，请向GitHub上的作者反馈。

红色的是正则表达式算子<https://zh.wikipedia.org/wiki/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F>

蓝色的单元格是未完成（目前无法完成）的内容。

值的分类<https://zh.wikipedia.org/wiki/%E5%80%BC_(%E9%9B%BB%E8%85%A6%E7%A7%91%E5%AD%B8)>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 广义左值  gvalue |  | 左值  lvalue | 变量名与函数名（即有名字的右值的引用）、先增、先减、解引用、赋值、复合赋值、下标（除了数组临终值）、左端操作数为左值的指针运算符、右端的操作数为左值的逗号运算符、第二与第三操作数都为左值的三元条件运算符 | 可以取地址 | 具有标识 | 可以取地址的表达式。  非临时对象。 |
| 右值  rvalue | 临终值  xvalue | 在c++11中引入  指向数据成员的指针表达式的第1操作数 | 不可以取地址 | 对应的对象接近生存期结束，但其内容尚未被移走。 |
|  | 纯右值  prvalue | 字面量（除了字符串字面量，字符串字面量是const左值）、函数调用、后增、后减、算术与逻辑运算符、比较运算符、取地址运算符、右端操作数为右值的逗号运算符、第二或第三操作数不是左值的三元条件运算符、数据类型转换表达式 | 不具有标识 | 对应临时对象或不对应任何对象的值。 |

运算符用法、功能、优先级和结合顺序（！！！运算符执行时至少求1个表达式的值。表达式的求值过程可能有side-effect，例如：调用函数进而变量的值被修改。因此是否去求表达式的值以及求多少次也都需要被关注。）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ()括号 |  | 改变运算符的结合顺序 |  |  |
| ()函数调用运算符 | **表达式1**((**表达式2**(,**表达式**)\*)?) | 调用1个函数。  返回值数据类型不为void的函数返回1个值。 | 1、表达式1是函数代码的起始地址，例如：函数名。  2、这里的逗号不是逗号运算符。 | 从左向右 |
| []数组下标 | **表达式1**[**表达式2**] | 得到数组的1个元素。 | 1、等价于\*(**表达式1**+**表达式2**)。  2、表达式1是数组的1个元素的地址，例如：数组名（数组首元素的地址）。  3、表达式2是(访问的元素的索引-表达式1对应的元素的索引)，数据类型是整型。 |
| .通过对象选择成员 | **表达式**.**成员名** | 得到对象的1个成员。 |  |
| ->通过指针选择成员 | **表达式**->**成员名** | 得到对象的1个成员。 | 1、表达式是对象的地址 |
| ++一元后置递增 | **表达式**++ | 得到表达式的值。  当前语句执行完后使表达式的值加1。 | ！！！得到的值是右值。  1、表达式的值是左值，访问限定没有设置const。 |
| --一元后置递减 | **表达式**-- | 得到表达式的值。  当前语句执行完后使表达式的值减1。 | ！！！得到的值是右值。  1、表达式的值是左值，访问限定没有设置const。 |
| ++一元前置递增 | ++**表达式** | 得到表达式的值。  当前语句执行之前使表达式的值加1。 | ！！！得到的值是左值。 | 从右向左 |
| --一元前置递减 | --**表达式** | 得到表达式的值。  当前语句执行之前使表达式的值减1。 | ！！！得到的值是左值。 |
| +一元正 | +**表达式** |  |  |
| -一元负 | -**表达式** |  |  |
| !一元逻辑非 | !**表达式** | 表达式的值等于0，得到1；表达式的值不等于0，得到0。 |  |
| ~一元按位取补 | ~**表达式** | 得到表达式的值中所有二进制位取反后的数。 | 1、表达式的数据类型是整型。 |
| (**类型名**)C风格的一元强制（数据）类型转换 | (**类型名**)**表达式** | 为操作数创建1个目标数据类型的临时副本。 | 1、浮点数转整数，舍去小数部分。  2、void数据类型的值不能转换为其它数据类型的值。 |
| \*解引用 | \***表达式** | 得到操作数（是1个地址）指向的对象。 | 1、得到的值是左值。  2、表达式的数据类型不是void\*。 |
| &取地址 | &**表达式** | 得到操作数的地址。 | 1、表达式的值是左值。  2、表达式不是register变量。 |
| sizeof按字节确定大小 | sizeof(**数据类型名**)  sizeof(**表达式**) | 例外：若表达式是数组名且该语句和数组声明在同一个程序块，得到数组占的字节数。 | 1、sizeof(**数组名**+0)得到指针数据类型占的字节数（4）。 |
| \*乘 | **表达式**\***表达式** |  |  | 从左向右 |
| /除以 | **表达式1**/**表达式2** |  | 1、表达式2的值不等于0。 |
| %求模 | **表达式1**%**表达式2** |  |
| +加 | **表达式**+**表达式** |  |  | 从左向右 |
| -减 | **表达式**-**表达式** |  |  |
| <<按位左移 | **表达式**1<<**表达式**2 | 得到表达式1的值按位向左移动表达式2位后的数。 | 1、右边腾空的数位补0。 | 从左向右 |
| >>按位右移 | **表达式**1>>**表达式**2 | 得到表达式1的值按位向右移动表达式2位后的数。 | 1、左边腾空的数位的填补方式取决于计算机。 |
| <小于 | **表达式**<**表达式** | 成立，得到1；不成立，得到0。 |  | 从左向右 |
| <=小于或等于 | **表达式**<=**表达式** |  |
| >大于 | **表达式**>**表达式** |  |
| >=大于或等于 | **表达式**>=**表达式** |  |
| ==关系等于 | **表达式**==**表达式** |  | 从左向右 |
| !=关系不等于 | **表达式**!=**表达式** |  |
| &按位与 | **表达式**&**表达式** | 对两个表达式的值进行逐位比较。相应的位都等于1，运算结果的相应的位为1；否则为0。 | 1、表达式的数据类型都是整型。 | 从左向右 |
| ^按位异或 | **表达式**^**表达式** | 对两个表达式的值进行逐位比较。相应的位不相等，运算结果的相应的位为1；否则为0。 | 1、表达式的数据类型都是整型。 |
| |按位同或 | **表达式**|**表达式** | 对两个表达式的值进行逐位比较。相应的位都等于0，运算结果的相应的位为0；否则为1。 | 1、表达式的数据类型都是整型。 |
| &&逻辑与 | **表达式1**&&**表达式2** | 表达式的值都不等于0，得到1；否则得到0。 | 1、先求表达式1的值。如果等于0，那么不求表达式2的值，得到0。 | 从左向右 |
| ||逻辑或 | **表达式1**||**表达式2** | 表达式的值都等于0，得到0；否则得到1。 | 1、先求表达式1的值。如果等于1，那么不求表达式2的值，得到1。 |
| ?:三元条件运算 | **表达式1**?**表达式2**:**表达式3** | 表达式1的值不等于0，得到表达式2的值；否则得到表达式3的值。 | 1、先求表达式1的值。如果等于1，那么不求表达式3的值，得到表达式2的值；如果等于0，那么不求表达式2的值，得到表达式3的值。 | 从右向左 |
| =赋值 | **表达式1**=**表达式2** |  |  | 从右向左 |
| +=加赋值 | **表达式1**+=**表达式2** |  |  |
| -=减赋值 | **表达式1**-=**表达式2** |  |  |
| \*=乘赋值 | **表达式1**\*=**表达式2** |  |  |
| /=除以赋值 | **表达式1**/=**表达式2** |  | 1、表达式2的值不等于0。 |
| %=取模赋值 | **表达式1**%=**表达式2** |  |
| &=按位与赋值 | **表达式1**&=**表达式2** |  |  |
| ^=按位异或赋值 | **表达式1**^=**表达式2** |  |  |
| |=按位同或赋值 | **表达式1**|=**表达式2** |  |  |
| <<=按位左移赋值 | **表达式1**<<=**表达式2** |  |  |
| >>=按位右移赋值 | **表达式1**>>=**表达式2** |  |  |
| ,逗号 | **表达式1**,**表达式2** | 得到表达式2的值。 | 1、先求表达式1的值，后求表达式2的值。 | 从左向右 |

运算符的操作数的定值顺序

|  |  |
| --- | --- |
| 逻辑与&&、逻辑或||、逗号运算符, | 从左向右 |
| 条件运算符?: | 先左，后中（左不等于0）或右（左等于0） |
| 其它二元运算符 | 未定义 |

标识符作用域

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数作用域 | 标号 | 在函数内定义 | 在函数中被访问 |  |
| 文件作用域 | 全局变量、函数定义、函数外的函数原型、typedef | 在所有函数外定义 | 从声明到程序结束被访问 | 1、不能出现两个标识符重名且作用域结束位置相同。  2、内外标识符重名时，在且仅在**内标识符**能被访问的区域（从声明到程序结束被访问/从声明到**当前**程序块结束被访问）**外标识符**不能被访问。 |
| 程序块作用域 | 局部变量、函数的形参、函数内的函数原型、typedef | 在程序块内定义 | 从声明到**当前**程序块结束被访问 |
| 函数原型作用域 | 函数原型的形参名 | 在函数原型内定义 |  |  |

标识符存储类型（！！！存储类型修饰符至多使用1个）（**变量定义语句**不初始化）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| auto | auto **变量定义语句**去掉**数据类型名**（必须初始化）<http://stackoverflow.com/questions/2192547/where-is-the-c-auto-keyword-used>  **变量定义语句** | 在程序块内定义 | 从声明到程序块执行结束被访问 | 变量在函数调用结束后释放。 | 1、不指定数据类型的根据初始值自动确定变量的数据类型。   |  | | --- | | void f(void){}  int main()  {  auto a=f;  } |   2、赋值之前变量的值未定义。 |
| register | register **变量定义语句** | 将变量驻留在寄存器中。  变量在函数调用结束后释放。 | 1、赋值之前变量的值未定义。  2、不能取变量的地址。  3、编译器可能忽略register。  4、编译器可能将auto变为register。 |
| extern | extern **变量定义语句**  extern **函数原型**  extern **函数定义** | 在程序块内或所有函数外定义 | 从声明到执行结束被访问 | 变量、函数在本文件后面定义或在另外的文件中定义。  变量在执行结束后被释放。 |  |
| static | static **变量定义语句**  static **函数原型**  static **函数定义** | 局部变量，函数执行结束后仍然保留。  全局变量，仅在定义它的文件中被访问。  变量在执行结束后被释放。 | 1、没有初始化相当于初始化为零。  2、至多初始化1次。 |
| typedef | typedef **变量定义语句**中**变量名**换成**新的数据类型名** |  | 创建数据类型的别名 | 存储类型不是数据类型的一部分，因此创建数据类型别名时不指定存储类型。 |

标识符存储周期

|  |  |  |
| --- | --- | --- |
| 自动存储周期 | 没链接 | 自动局部变量（auto）、寄存器（局部）变量（register） |
| 静态存储周期 | 内部链接 | 静态局部变量（static）、static全局变量、static函数的名 |
| 外部链接 | 非static全局变量、非static函数的名 |

变量访问限制（！！！访问限定符可以使用多个）

|  |  |
| --- | --- |
| const | 定义后无法修改。无extern修饰的const变量仅在定义它的文件中被访问；有extern修饰的const变量可以被另外的文件访问。 |
| volatile | 变量随时有可能改变。因此编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中读取数据（而不是暂时使用寄存器中的值）。 |
| restrict | 只适用于对象指针类型。指针所指向的对象如果被修改，就不可以被此指针以外的方式所存取，不管是直接地还是间接地。 |

声明一个变量意味着向编译器描述变量的类型，但并不为变量分配存储空间。

定义一个变量意味着在声明变量的同时还要为变量分配存储空间。

基本数据类型

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  | | 32位系统 |  | | (signed)?/unsigned | char | 1 |  | | short( int)? | 2 |  | | int | 4 |  | | long( int)? | 4 |  | | long long | 8 |  | | float | | 4 |  | | double | | 8 |  | | long double | | 8 |  | | |
| （通过**基本数据类型**）定义变量  (**访问限定符** )\***基本数据类型名** **变量名**(=**表达式**)?(,**变量名**(=**表达式**)?)\*; | 1、不能定义void类型的变量。  2、变量的数据类型是(**访问限定符** )\***基本数据类型**。  3、若初始化，表达式的数据类型去掉所有访问限定符是**基本数据类型**。  4、若访问限定符列表中有const，则必须初始化。  5、const **基本数据类型名**和**基本数据类型名** const等价。 |
| 溢出   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 整数溢出   |  |  | | --- | --- | | unsigned整型 | 溢出后的数以2^(8\*sizeof(type))作模运算。 | | signed整型 | 未定义 | | | 浮点数溢出（infinity, indeterminate）   |  |  | | --- | --- | | 1.#INF(0)\* | 自己写的数（得到正无穷）、正数/零（得到正无穷）、正无穷加减常数、正无穷加正无穷、正无穷乘正数、负无穷乘负数 | | -1.#INF(0)\* | 自己写的数（得到负无穷）、负数/零（得到负无穷）、负无穷加减常数、负无穷减负无穷、正无穷乘负数、负无穷乘正数 | | -1.#IND(0)\* | 零/零、零\*无穷大、无穷大/无穷大、无穷大-无穷大、自变量∉定义域 | | | |

派生数据类型

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| （**某数据类型**的）指针   |  |  |  | | --- | --- | --- | | 定义变量  在**某数据类型**的变量定义语句中将**变量名**换为\*(const )?**变量名**，如果变量名后紧接着[则需要加1层括号将\*和变量名括起来（因为[]优先级更高）。  例外：可以定义void\*(const)?数据类型的变量。 | | ！！！新数据类型还可以继续用作**某数据类型**。  1、新数据类型是在**某数据类型**最后1个\*(const )?后（如果没有\*(const )?则是第1个[前）插入1个\*，如果插入的\*后紧接着[则需要加1层括号将\*括起来（因为[]优先级更高）。新数据类型占4个字节。  2、若初始化，表达式的数据类型去掉所有访问限定符与**某数据类型**去掉所有访问限定符相同。  3、若最后一个\*后有const，则必须初始化。 | | 1、\*后有const表示不能通过该变量解引用得到的值修改数据。（仍然可能可以通过其他途径修改数据。）   |  | | --- | | const int \*const \* \*const p=NULL;  p=NULL;错误，不能修改常量  \*p=NULL;正确  \*\*p=NULL;错误，不能修改常量  \*\*\*p=0;错误，不能修改常量 | | | 操作  =赋值  \*解引用  &取地址  sizeof按字节确定大小  ->通过指针选择成员  +加  -减  +=加赋值  -=减赋值  ++一元后置递增  --一元后置递减  ++一元前置递增  --一元前置递减  []数组下标 | **指针**+**表达式**  **表达式**+**指针**  **指针**-**指针**  **表达式1**=**表达式2** | 1、数据类型为void\*的值不能进行和加减有关的操作，不能解引用。  2、数据类型不为void\*的值赋给数据类型为void\*的变量不需要强制（数据）类型转换运算符。  3、能不通过强制（数据）类型转换运算符就赋给指针的常量只有0和NULL。  4、表达式的数据类型是整型。  5、**表达式2**的数据类型中所有有const的位置，**表达式1**的数据类型中的所有相应位置都有const。最右侧的\*是否有const无所谓。忽略最右侧的\*和**表达式2**中有const的\*后，若1个\*有const，其右侧的所有\*都有const。（最左侧的基本数据类型也当作1个\*处理。）（若1个\*没有const，其左侧的所有\*都没有const。）<http://c-faq.com/ansi/constmismatch.html>例（不使用强制（数据）类型转换运算符）：   |  | | --- | | int \* \*const \* \* \* \* a=0;  int \* \*const \* \* \* \* \*b1=&a;  int \* \*const \* \* \* \*const \*b2=&a;  int \* \*const \* \* \*const \*const \*b3=&a;  int \* \*const \* \*const \*const \*const \*b4=&a;  int \* \*const \*const \*const \*const \*const \*b5=&a;  int \*const \*const \*const \*const \*const \*const \*b6=&a;  const int \*const \*const \*const \*const \*const \*const \*b7=&a;  int \* \*const \* \* \* \* \*const c1=&a;  int \* \*const \* \* \* \*const \*const c2=&a;  int \* \*const \* \* \*const \*const \*const c3=&a;  int \* \*const \* \*const \*const \*const \*const c4=&a;  int \* \*const \*const \*const \*const \*const \*const c5=&a;  int \*const \*const \*const \*const \*const \*const \*const c6=&a;  const int \*const \*const \*const \*const \*const \*const \*const c7=&a; | | int \* \*const \* \*const \* \* a=0;  int \* \*const \* \*const \* \* \*b1=&a;  int \* \*const \* \*const \* \*const \*b2=&a;  int \* \*const \* \*const \*const \*const \*b3=&a;  int \* \*const \*const \*const \*const \*const \*b4=&a;  int \*const \*const \*const \*const \*const \*const \*b5=&a;  const int \*const \*const \*const \*const \*const \*const \*b6=&a;  int \* \*const \* \*const \* \* \*const c1=&a;  int \* \*const \* \*const \* \*const \*const c2=&a;  int \* \*const \* \*const \*const \*const \*const c3=&a;  int \* \*const \*const \*const \*const \*const \*const c4=&a;  int \*const \*const \*const \*const \*const \*const \*const c5=&a;  const int \*const \*const \*const \*const \*const \*const \*const c6=&a; |   6、语言不负责内存边界检查。 | | 1、指针+整数得到的值是指针的值+整数\*sizeof(**某数据类型**)。  2、指针-指针得到的值是(指针的值-指针的值)/sizeof(**某数据类型**)。两个指针指向的变量是同一数组的元素时才有意义。  3、取地址**新数据类型**的变量=&**某数据类型**的变量;。  4、**新数据类型**的变量->成员名等价于(\***新数据类型**的变量).成员名。  5、\*p+=1等价于(\*p)++。 | |
| （**某数据类型**的）数组   |  |  | | --- | --- | | 定义变量  在**某数据类型**的变量定义语句中将**变量名**换为**变量名**[**元素个数**]，将(=**表达式**)?换为(={(**表达式**(,**表达式**)\*)?)?}  或  将**变量名**换为**变量名**[]，将(=**表达式**)?换为={**表达式**(,**表达式**)\*} 。 | ！！！新数据类型还可以继续用作**某数据类型**。  1、新数据类型是在**某数据类型**最后1个\*(const )?后（如果没有\*(const )?则是第1个[前）插入1个[**元素个数**]。新数据类型占sizeof(**某数据类型**)\***元素个数**个字节。  2、若初始化，表达式的数据类型去掉所有访问限定符与**某数据类型**去掉所有访问限定符相同。  3、若最后一个\*后有const，则必须初始化。  4、若初始化且初始值个数小于成员个数，剩余元素：非指针非数组初始化为0，指针初始化为NULL。 | | 1、char s[]="(**字符**)\*";等价于char s[]={('**字符**',)\*'\0'};。 | | 操作  \*解引用  &取地址  sizeof按字节确定大小  []数组下标 | 1、数组下标的取值从0到**元素个数**-1。  2、语言不负责内存边界检查。 | | 1、**变量名**[**下标**]等价于\*(**变量名**+**下标**)。  2、&**变量名**的数据类型是在**某数据类型**最后1个\*后（如果没有\*则是第1个[前）插入1个\*，占4个字节。<http://blog.csdn.net/daniel_ice/article/details/6857019>   |  | | --- | | int c[3][5],(\*p1)[5],(\*p2)[3][5];  p1=c;p2=&c;  printf("%p %p %d\n%p %p %d\n%p %p %d\n%p %p %d\n"  ,c,c+1,(int)(c+1)-(int)c  ,&c,&c+1,(int)(&c+1)-(int)&c  ,p1,p1+1,(int)(p1+1)-(int)p1  ,p2,p2+1,(int)(p2+1)-(int)p2);  结果：c、p1各增加20，&c、p1各增加60。 | | |
| 判断变量是指针还是数组：看最靠近**变量名**的符号  ...\*(const )?**变量名**)，指针  (**变量名**[元素个数]...，数组  ...\*(const )?**变量名**[元素个数]...，数组  ！！！再强调一遍，所有的\*后都可以带1个const，若最后一个\*后有const则必须初始化。（下表中的类型都指数据类型。）   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | int v;  (int)  1个int类型的变量 | | | | | | | | | 的指针 | | | | 的含3个该类型的元素的数组 | | | | | int\*v;  (int\*)  1个int\*类型的变量（1个储存int类型变量地址的变量） | | | | int v[3];  (int[3])  1个int[3]类型的变量（3个int类型的变量） | | | | | 的指针 | | 的含3个该类型的元素的数组 | | 的指针 | | 的含5个该类型的元素的数组 | | | int\*\*v;  (int\*\*)  1个int\*\*类型的变量（1个储存int\*类型变量地址的变量） | | int\*v[3];  (int\*[3])  1个int\*[3]类型的变量（3个int\*类型的变量） | | int(\*v)[3];  (int(\*)[3])  1个int(\*)[3]类型的变量（1个储存int[3]类型变量地址的变量） | | int v[5][3];  (int[5][3])  1个int[5][3]类型的变量（5个int[3]类型的变量） | | | 的指针 | 的含3个该类型的元素的数组 | 的指针 | 的含5个该类型的元素的数组 | 的指针 | 的含5个该类型的元素的数组 | 的指针 | 的含7个该类型的元素的数组 | | int\*\*\*v;  (int\*\*\*)  1个int\*\*\*类型的变量（1个储存int\*\*类型变量地址的变量） | int\*\*v[3];  (int\*\*[3])  1个int\*\*[3]类型的变量（3个int\*\*类型的变量） | int\*(\*v)[3];  (int\*(\*)[3])  1个int\*(\*)[3]类型的变量（1个储存int\*[3]类型变量地址的变量） | int\*v[5][3];  (int\*[5][3])  1个int\*[5][3]类型的变量（5个int\*[3]类型的变量） | int(\*\*v)[3];  (int(\*\*)[3])  1个int(\*\*)[3]类型的变量（1个储存int(\*)[3]类型变量地址的变量） | int(\*v[5])[3];  (int(\*[5])[3])  1个int(\*[5])[3]类型的变量（5个int(\*)[3]类型的变量） | int(\*v)[5][3];  (int(\*)[5][3])  1个int(\*)[5][3]类型的变量（1个储存int[5][3]类型变量地址的变量） | int v[7][5][3];  (int[7][5][3])  1个int[7][5][3]类型的变量（7个int[5][3]类型的变量） | |

自定义数据类型

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 结构体   |  |  |  | | --- | --- | --- | | 声明数据类型  struct( **数据类型名**)?  {  (**变量定义语句**  )\*  }; | 从声明到程序结束被访问/从声明到程序块结束被访问。  从声明到执行结束被访问/从声明到程序块执行结束被访问。 | 1、成员变量的存储类型为auto或static。若在所有函数外声明，当且仅当存储类型为static时，访问限定设置const，必须初始化（没初始化则不可以访问），不计入sizeof(struct **数据类型名**)和sizeof(**变量名**)；若在程序块内声明，存储类型不可以为static。访问限定设置const时，变量的数据类型是整型。  2、成员的数据类型不是当前正在声明的数据类型。（但可以是当前正在声明的数据类型的派生数据类型，派生关系含有至少1个\*关系（即不能光有[]关系）。因为\*关系会把成员数据类型变成**某数据类型**的指针，大小由未知转为已知或保持已知。例如：  struct **数据类型名** **成员名**[1][1][1][1];错误，  struct **数据类型名** (\***成员名**[1][1])[1][1];正确。）  3、0长度数组成员<http://coolshell.cn/articles/11377.html>。 | | 定义变量  (**访问限定符** )\*struct **数据类型名 变量名**(={(**表达式**(,**表达式**)\*)?})?; | | 1、若初始化且初始值个数小于成员个数，剩余成员：非指针初始化为0，指针初始化为NULL。  2、如果数据类型在程序块内声明且包含访问限定设置const的成员，必须初始化。 | | 操作  =赋值  &取地址  .访问成员  sizeof按字节确定大小 | | 1、不能用==、!=比较两个结构体。  2、赋值操作也会复制数组成员的每一个元素。  3、含有[指向动态分配的内存的指针]的结构体的赋值操作会有浅拷贝的问题。 | | 在声明数据类型时定义变量  struct( **数据类型名**)?  {  (**变量定义语句**  )\*  }**变量定义语句**; | | 1、大括号后的变量定义语句为正常变量定义语句去掉struct **数据类型名**。  2、大括号后的变量定义语句的每个访问限定符都可以移动到struct前。  3、若指定存储类型，存储类型修饰符可以在struct前或在变量定义语句前。 | | 在声明数据类型时使用typedef  typedef struct( **数据类型名**)?  {  (**变量定义语句**  )\*  }**数据类型名**(,**数据类型名**)\*; | |  | |
| 共用体   |  |  |  | | --- | --- | --- | | 声明数据类型  union( 数据**类型名**)?  {  (**变量定义语句**  )\*  }; | 从声明到程序结束被访问/从声明到程序块结束被访问。  从声明到执行结束被访问/从声明到程序块执行结束被访问。 |  | | 定义变量  (**访问限定符** )\*union **数据类型名 变量名**(={(**表达式**)?})?; | | 1、若表达式的值的数据类型与第1个成员的数据类型不同，则表达式的值被转换为第1个成员的数据类型的值。浮点数转整数，舍去小数部分。 | | 操作  =赋值  &取地址  .访问成员  sizeof按字节确定大小 | | 1、不能用==、!=比较两个共用体。 | | 在声明数据类型时定义变量  与结构体类似。 | |  | | 在声明数据类型时使用typedef  与结构体类似。 | |  | |
| 枚举（数据）类型   |  |  |  | | --- | --- | --- | | 声明  enum **数据类型名**{(**标识符**(=**表达式**)?(,**标识符**(=**表达式**)?)\*)?}; | 从声明到程序结束被访问/从声明到程序块结束被访问。  从声明到执行结束被访问/从声明到程序块执行结束被访问。 | 1、表达式数据类型是整型，不含有变量、函数调用。  2、不指定第1个标识符的值相当于指定为0。除了第1个标识符，没有指定值的标识符的值等于前1个的值加1。 | | 定义变量  (**访问限定符** )\*enum **数据类型名 变量名**(=**标识符**)?;  (**访问限定符** )\*enum **数据类型名** **变量名**=(enum **数据类型名**)**表达式**; | | 1、标识符不能使用其它enum数据类型的标识符。  2、表达式的值可以超出enum数据类型含有的标识符的值的范围。  3、表达式的值的数据类型为该数据类型时，可以省略强制（数据）类型转换运算符。 | | 在声明数据类型时定义变量  与结构体类似。 | |  | | 在声明数据类型时使用typedef  与结构体类似。 | |  | |

创建数据类型的别名

|  |
| --- |
| 创建别名  typedef **变量定义语句**中**变量名**换成**新的数据类型名**  **变量定义语句**不初始化。 |
| （使用别名）定义变量  (**访问限定符** )\***新的数据类型名** **变量名**(=**表达式**)?(,**变量名**(=**表达式**)?)\*; |

控制转移

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 选择结构   |  |  |  | | --- | --- | --- | | if(**表达式**)  {  **语句**  } | 1、语句外的大括号，当且仅当语句数等于1时，可以省略。 |  | | if (**表达式**)  {  **语句**  }  else  {  **语句**  } | 1、语句外的大括号，当且仅当语句数等于1时，可以省略。 |  | | switch(**表达式**)  {  (case **表达式**:  **语句**  )+} | 1、switch的表达式的数据类型是整型。  2、case的表达式的数据类型是整型，不含有变量、函数调用。1个switch语句直接包含的每个case后表达式的值互不相同。  3、case **表达式**:可以换成default:，1个switch语句直接包含至多1个default:，直接包含至少1个case。  4、（无论default在什么位置，）若无case被匹配，控制跳到default:。 | 1、1个case的语句的最后1句为break;可以防止执行后面所有并列的case。 | |
| 循环结构   |  |  | | --- | --- | | while(**表达式**)  {  **语句**  } | 1、语句外的大括号，当且仅当语句数等于1时，可以省略。 | | for((**语句**)?;(**表达式**)?;(**语句**)?)  {  **语句**  } | 相当于   |  | | --- | | for后括号内第1条**语句**  while(**表达式**)  {  **语句**  for后括号内第2条**语句**  } |   1、语句外的大括号，当且仅当语句数等于1时，可以省略。  2、for后的括号内不出现其它的分号。  3、for后的括号内表达式不写相当于写1。 | | do  {  **语句**  }while(**表达式**); | 1、语句外的大括号，当且仅当语句数等于1时，可以省略。 | |
| 跳转   |  |  |  | | --- | --- | --- | | break; | 在switch中，跳过剩余语句；在while、for、do中，退出循环。 | 1、只能在switch、while、for、do中使用  2、只能退出直接包含该语句的1层switch/while/for/do。 | | continue; | 略过循环体剩余语句1次。 | 1、只能在while、for、do中使用。  2、只能略直接包含该语句的1层while/for/do。 | | return( **表达式**)?; | 将控制返回给主调函数。  在返回值的数据类型不为void的函数中，返回表达式的值。 | 1、返回值的数据类型不为void的函数必须包含至少1个return语句。  2、返回值的数据类型为void的函数也可以return(void)**表达式**;。 | | goto **标号**; |  | 1、default是关键字，不能做标识符，因此不能做标号。 | |
| 非本地跳转  #include<setjmp.h>   |  | | --- | | int setjmp(jmp\_buf env);  不是因longjmp被调用而调用，返回0；因longjmp被调用而调用：若longjmp的val不等于0，返回val；若longjmp的val等于0，返回1。  访问限定没有设置volatile的auto变量的值不确定。 | | void longjmp(jmp\_buf env,int val);  恢复env保存的调用环境，最后1次修改env的setjmp调用语句得到1个值（若val不等于0，返回val；若val等于0，返回1）。  env没有被setjmp初始化或最后1次修改env的setjmp所在的函数已经执行结束，未定义。 | |
| 其它程序结束方法  #include<stdlib.h>   |  | | --- | | int atexit(void(\*func)(void));  成功，返回0；失败，返回非0值。  程序结束时所有被注册函数的调用顺序为注册顺序的相反顺序。  （1个函数可以被注册多次。）  注册之后无法撤销。  注册的函数的个数可能有1个上限（不小于32）。  若atexit在exit后调用，未定义（和库的实现方法、系统有关）。 | | void exit(int status);   |  | | --- | | 调用atexit注册的所有函数。 | | 所有流被关闭（有缓冲区的，缓冲区被清空）。tmpfile创建的所有文件被删除。 | | 控制返回宿主环境。 |   status   |  |  | | --- | --- | | EXIT\_SUCCESS或0 | 返回1个代表程序成功结束的数值。 | | EXIT\_FAILURE | 返回1个代表程序未成功结束的数值。 | | 其它值 | （和库的实现方法、系统有关。） | | |

函数

|  |  |
| --- | --- |
| 定义  **返回值数据类型** **函数名**()  {  **语句**  }  **返回值数据类型** **函数名**(void)  {  **语句**  }  **返回值数据类型** **函数名**(**变量定义语句**(,**变量定义语句**)\*(,...)?)  {  **语句**  }  每个**变量定义语句**只定义1个变量。  去掉**变量定义语句**末尾的分号。  **变量定义语句**不初始化。 | 1、函数名后()表示允许传递任意实参；函数名后(void)表示不接收实参。  2、返回值的数据类型不为数组。  3、不返回值的函数，返回值的数据类型为void。  4、数组的第1维（对应的元素占的字节数最多的1维），长度可以不写，写的必须是正整数，会被忽略。 |
| 1、函数定义的第一行是**函数头**。  2、可变长的实参列表#include<stdarg.h><http://blog.csdn.net/jackystudio/article/details/17523523>   |  | | --- | | va\_list ap;  va\_start(ap,**最后1个普通形参名**);  (va\_arg(ap,**数据类型名**);)\* ←一般用循环实现  va\_end(ap); |   调用va\_arg会修改va\_list对象，使其指向下一个形参。  va\_end必须在va\_arg读完所有参数后再调用，否则会产生意想不到的后果。  由于没有原型声明，主调函数对每个参数进行数据类型提升：float数据类型的实际参数将提升到double；char、short和相应的signed、unsigned数据类型的实际参数提升到int；如果int不能存储原值，则提升到unsigned int。 |
| 声明函数原型  **函数头**; | 1、函数原型内形参名可以不写，可以写且与函数定义不一致。写的必须合法且不重复。 |
|  |
| 调用  **函数名**()  **函数名**(**形参名**(,**形参名**)\*) | 1、返回值的数据类型不为void的函数返回1个值。  2、数据类型提升规则：在没有函数原型的情况下，char与short数据类型都将被转换为int数据类型，float数据类型将被转换为double数据类型。 |
|  |
| 函数指针   |  |  |  | | --- | --- | --- | | 定义  **返回值数据类型** (\***变量名**)()(=**表达式**)?;  **返回值数据类型** (\***变量名**)(void)(=**表达式**)?;  **返回值数据类型** (\***变量名**)(**变量定义语句**(,**变量定义语句**)\*(,...)?)(=**表达式**)?;  每个**变量定义语句**只定义1个变量。  去掉**变量定义语句**末尾的分号。  **变量定义语句**不初始化。 | | 1、形参名可以不写。写的必须合法且不重复。  2、**表达式**是函数代码的起始地址，例如：**函数名**、&**函数名**、(\*)+**函数名**。 | | 1、**返回值数据类型** **变量名**(**形参列表**)=**函数名**;和**返回值数据类型** **变量名**(**形参列表**)=\***函数名**;是错误的，不存在将1个值转换为函数数据类型的值的转换过程。数据类型一样也不行：   |  | | --- | | int f(int a){return a;}  int main()  {  int a(int);  int b(int)=\*a;错误，不能把int(int)数据类型的值转换为int(int)数据类型的值  int b(int)=\*f;错误，不能把int(int)数据类型的值转换为int(int)数据类型的值  } |   2、**变量名**的数据类型是**返回值数据类型** (\_\_cdecl \*)(**形参数据类型列表**)，占4个字节；  &**函数名**的数据类型是**返回值数据类型** (\_\_cdecl \*)(**形参数据类型列表**)，占4个字节；  **函数名**的数据类型是**返回值数据类型** (\_\_cdecl \*)(**形参数据类型列表**)，不能使用sizeof；  (\*)+**函数名**的数据类型是**返回值数据类型** (**形参数据类型列表**)，不能使用sizeof。  <http://www.cnblogs.com/chinazhangjie/archive/2012/08/18/2645475.html> | | 操作  =赋值  ()函数调用运算符  \*解引用  &取地址  sizeof按字节确定大小 | (\***函数指针**)()  (\***函数指针**)(**形参名**(,**形参名**)\*) | 1、能赋给指针的常量只有0和NULL。  2、(\***函数指针**)可以换成**函数指针**。 | | 1、(\*)+**函数指针**的数据类型为**返回值数据类型** (\_\_cdecl \*)(**形参数据类型列表**)。  2、&**函数指针**的数据类型为**返回值数据类型** (\_\_cdecl \*\*)(**形参数据类型列表**)。  3、**返回值数据类型** (\_\_cdecl \*)(**形参数据类型列表**)、**返回值数据类型** (\_\_cdecl \*\*)(**形参数据类型列表**)、**返回值数据类型** (**形参数据类型列表**)数据类型的值都可以转换为void\*数据类型的值，但没有任何通过void\*数据类型的值调用函数的方法（因为不存在将1个值转换为函数数据类型的值的转换过程）。 | | 函数形参的数据类型是函数指针  **变量定义语句**换成**指向的函数的返回值数据类型** (\***形参名**)(**指向的函数的形参列表**)。 | | 1、指向的函数的形参列表里的形参名可以不写。写的必须合法且不重复。  2、(\***形参名**)可以换成**形参名**。 | |  | | 函数返回值的数据类型是函数指针  **返回值数据类型** **函数名**(**形参列表**)换成  **指向的函数的返回值数据类型**(\***函数名**(**形参列表**))(**指向的函数的形参列表**)。 | | 1、指向的函数的形参列表里的形参名可以不写。写的必须合法且不重复。 | |  | | |
| 判断函数出错原因  #include<errno.h>   |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | errno宏  值是左值，值的数据类型为int。  程序开始执行时值为0。  使用方法：在某函数调用前将errno设置为0，调用后读取errno的值。   |  |  |  | | --- | --- | --- | | EDOM | 参数的值超出数学函数的定义域。 | | | ERANGE | 返回的值超出返回值数据类型所能表示的范围。 | 负无穷、正无穷<https://zh.wikipedia.org/wiki/%E6%93%B4%E5%B1%95%E5%AF%A6%E6%95%B8%E7%B7%9A>  （经常在参数等于定义区间的端点时出现。） | | 下溢、上溢（整数，指小于最小值、大于最大值；浮点数，指量级太小、太大。） | | EILSEQ | 宽字符，字符顺序不合法。 | | | ... | ... | | | | |

操纵内存的函数

#include<stdlib.h>

|  |
| --- |
| void\*malloc(size\_t size);  成功，返回分配到的内存空间的首地址；失败，返回NULL。  分配到的内存空间是未初始化的。 |
| void\*calloc(size\_t num,size\_t size);  成功，返回分配到的内存空间的首地址；失败，返回NULL。  分配到的内存空间所有字节被初始化为0。  size等于0，未定义（和库的实现方法有关）。 |
| void\*realloc(void\*ptr,size\_t size);   |  |  |  | | --- | --- | --- | |  | C90 | C99、C11 | | 成功 | 返回分配到的内存空间的首地址  若新内存空间大小比旧内存空间大小大，超出部分内容不确定。  （旧内存空间上的内容被复制到新内存空间上，旧内存空间被释放。） | | | size等于0 | 返回NULL。  ptr指向的内存空间被释放。 | 未定义（可能返回NULL或不能被解引用的地址）。 | | 失败 | 返回NULL。  （ptr指向的内存空间不受任何影响。） | |   ptr等于NULL，相当于malloc(size);。 |
| void free(void\*ptr);  若ptr不指向malloc、calloc或realloc分配的内存且不等于NULL，未定义；若ptr等于NULL，不造成任何影响。 |

#include<string.h>

|  |
| --- |
| void\*memset(void\*ptr,int value,size\_t num);  返回ptr的值。 |
| void\*memcpy(void\*destination,const void\*source,size\_t num);  返回destination的值。  destination指向的num个字节的内存空间和source指向的num个字节的内存空间有重叠，未定义。 |
| void\*memmove(void\*destination,const void\*source,size\_t num);  返回destination的值。  （destination指向的num个字节的内存空间和source指向的num个字节的内存空间可以有重叠。） |
| int memcmp(const void\*ptr1,const void\*ptr2,size\_t num);  停止时ptr1的字节比ptr2小，返回-1；停止时s1的字节和s2相等，返回0；停止时s1的字节比s2大，返回1。  （字节不相等或比较num个字节，停止。）  （返回0说明比较的部分相等。） |
| void\*memchr(void\*ptr,int value,size\_t num);  返回ptr指向的num个字节的内存空间中第1个value的地址。 |

字符处理函数

#include<ctype.h><http://www.cplusplus.com/reference/cctype/isblank/>

|  |
| --- |
| int isxdigit(int c);  c是'0'~'9'、'a'~'f'、'A'~'F'，返回1；否则返回0。 |
| int isalnum(int c);  c是数字或字母，返回1；否则返回0。 |
| int isdigit(int c);  c是数字，返回1；否则返回0。 |
| int isalpha(int c);  c是字母，返回1；否则返回0。 |
| int islower(int c);  c是小写字母，返回1；否则返回0。 |
| int isupper(int c);  c是大写字母，返回1；否则返回0。 |
| int iscntrl(int c);  c是控制字符（'\a''\b''\f''\n''\r''\t''\v'），返回1；否则返回0。 |
| int isprint(int c);  c是（含空格在内的）可打印字符，返回1；否则返回0。 |
| int isgraph(int c);  c是（除空格以外的）可打印字符，返回1；否则返回0。 |
| int ispunct(int c);  c是（除空格、数字、字母以外的）可打印字符，返回1；否则返回0。 |
| int isspace(int c);  c是' ''\f''\n''\r''\t''\v'，返回1；否则返回0。 |
| int isblank(int c);  c是' ''\t'，返回1；否则返回0。 |
| int tolower(int c);  c是大写字母，将c转为小写字母返回；否则返回c。 |
| int toupper(int c);  c是小写字母，将c转为大写字母返回；否则返回c。 |

字符串转换函数

#include<stdlib.h>

|  |  |
| --- | --- |
| int atoi(const char\*nPtr); | long strtol(const char\*nPtr,char\*\*endPtr,int base);  成功，返回转换后的长整型数；不能转换或字符串为空字符串，返回0L；转换得到的值超出long int所能表示的范围，将errno的值设置为ERANGE，返回LONG\_MAX或LONG\_MIN（在limits.h中定义）。  endPtr指向1个char\*数据类型的变量，用来储存字符串转换成数字后余下部分的起始地址。  base表示进制数，取值0，2~36。  0：默认采用10进制转换，字符串以"0x""0X"开始，使用16进制转换；字符串以"0"开始，使用8进制转换。  11~36：用A~Z表示10~35。 |
| long atol(const char\*nPtr); | unsigned long strtoul(const char\*nPtr,char\*\*endPtr,int base);  成功，返回转换后的无符号长整型数；不能转换或字符串为空字符串，返回0；转换得到的值超出unsigned long int所能表示的范围，将errno的值设置为ERANGE，返回ULONG\_MAX（在limits.h中定义）。 |
| double atof(const char\*nPtr); | double strtod(const char\*nPtr,char\*\*endPtr);  成功，返回转换后的双精度浮点数；没有执行有效的转换，返回0.0。 |

字符串处理函数

#include<string.h>

|  |  |
| --- | --- |
| char\*strcpy(char\*s1,const char\*s2);  返回s1。 | 复制'\0'。 |
| char\*strncpy(char\*s1,const char\*s2,size\_t n);  返回s1。 | 遇到'\0'，以'\0'填充直至写入n个字符；复制n个字符，停止。  （不一定复制'\0'。） |
| char\*strcat(char\*s1,const char\*s2);  返回s1。 | （s1末尾'\0'被覆盖。） |
| char\*strncat(char\*s1,const char\*s2,size\_t n);  返回s1。 | 遇到'\0'，停止；读入n个字符，停止，末尾添加'\0'。  '\0'不计入n个字符内。（最多n+1个元素被修改。） |
| int strcmp(const char\*s1,const char\*s2);  停止时s1的字符比s2小，返回-1；停止时s1的字符和s2相等，返回0；停止时s1的字符比s2大，返回1。 | 遇到'\0'或字符不相等，停止。  （返回0说明s1==s2。） |
| int strncmp(const char\*s1,const char\*s2,size\_t n);  返回停止时s1的字符-停止时s2的字符。 | 遇到'\0'或字符不相等或比较n个字符，停止。  （返回0说明比较的部分相等。） |
| char\*strchr(const char\*s,int c);  返回s中第1个c的地址或NULL。 |  |
| char\*strrchr(const char\*s,int c);  返回s中最后1个c的地址或NULL。 |  |
| char\*strpbrk(const char\*s1,const char\*s2);  返回s1中第1个[s2中任意字符]的地址或NULL。 |  |
| char\*strstr(const char\*s1,const char\*s2);  返回s1中第1个s2的地址或NULL。 |  |
| size\_t strspn(const char\*s1,const char\*s2);  返回s1中只包含s2中字符起始片段长度。 | '\0'不计入内。 |
| size\_t strcspn(const char\*s1,const char\*s2);  返回s1中不包含s2中字符起始片段长度。 | '\0'不计入内。 |
| char\*strtok(char\*s1,const char\*s2);  调用产生新标号，返回当前标号地址；  调用不产生新标号，返回NULL。 | 第1次调用。strtok(待标号化的字符串,包含分隔符的字符串)。返回第1个不是分隔符的字符的地址（即第1个标号的地址），将第1个标号后第1个分隔符替换为'\0'。（保存该字符的下1个字符的地址。）  第≥2次调用。strtok(NULL,包含分隔符的字符串)。返回（从保存的地址开始往后）第1个不是分隔符的字符的地址（即第n个标号的地址）或NULL（没找到且遇到字符串结束符'\0'），将第n个标号后第1个分隔符替换为'\0'。（保存该字符的下1个字符的地址。） |
| char\*strerror(int errornum);  返回错误号对应的错误说明。 |  |
| size\_t strlen(const char\*s);  返回'\0'前的字符个数。 |  |

#include<stdio.h>

|  |  |
| --- | --- |
| int sscanf(const char\*s,const char\*format,...);  成功，返回被赋值的变量的个数（可能为0）；在读取所有数据之前出错，返回EOF。 | int sprintf(char\*s,const char\*format,...);  成功，返回输出字符个数（'\0'不计入内）；发生错误，返回一个负数。 |
| int snprintf(char\*s,size\_t n,const char\*format,...);  成功，返回s的字符个数（'\0'不计入内）；发生错误，返回一个负数。  至多打印n个字符（包括'\0'）。 |

操纵标准流的函数

#include<stdio.h>

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| int scanf(const char\*format,...);  成功，返回被赋值的变量的个数（可能为0）；发生错误，设置文件错误标志，返回EOF；遇到结束符，设置文件结束标志，返回EOF；在读取宽字符的时候发生编码错误，将errno设置为EILSEQ。  格式控制字符串组成：文本字符、转义序列、转换说明。  文本字符  空格表示忽略1串连续出现的广义空格（' '、'\t'、'\n'等）。  转义序列   |  |  |  |  |  | | --- | --- | --- | --- | --- | | \' |  | ' | | | | \" |  | " | | | | \? |  | ? | | | | \\ |  | \ | | | | \a | 告警或铃声 | 引发1个听得见的（铃声）或看得见的警报 | | | | \b | 退格 | 将光标 | 在当前行中向后移动1个位置 | | | \f | 新的一页或换页 | 移动到 | 新的一个逻辑页面的开始位置 | | \n | 新的一行 | 新的一行的开始位置 | | \r | 回车 | 当前行的开始位置 | | \t | 水平Tab | 下一个水平制表符位置 | | \v | 垂直Tab | 下一个垂直制表符位置 |   转换说明：%[\*][域宽][长度修饰符]转换说明符  赋值抑制字符：没有，正常；有，从输入流中读取数据并将其丢弃。  域宽：读取的字符个数。   |  |  |  |  | | --- | --- | --- | --- | | 长度修饰符 | 转换说明符 | 说明 | 实参数据类型 | | h短整型  l长整型 | d | 任意符号10进制整数 | 指向整型变量的指针 | | i | 任意符号10、8或16进制整数 | | o | 8进制整数 | 指向无符号整型变量的指针 | | u | 无符号10进制整数 | | x、X | 16进制整数 | | l双精度浮点数  L长双精度浮点数 | e、E、f、g、G | 浮点数 | 指向浮点数数据类型变量的指针 | |  | c | 字符 | 指向字符数据类型变量的指针 | | s | 字符串，自动加上'\0'。 | 指向字符数据类型变量的指针 | | [(**字符**)+] | 匹配，存储；不匹配，扫描结束。自动加上'\0'。 | 指向字符数据类型变量的指针 | | [^(**字符**)+] | 不匹配，存储；匹配，扫描结束。自动加上'\0'。 | | p | 地址 | 指向指针的指针 | | n | 保存到目前为止本次scanf已输入的字符总数 | 指向整型变量的指针 | | % | 忽略1个百分号 |  |   清理输入流   |  |  |  | | --- | --- | --- | | fflush(stdin); | while(getchar()!='\n'); | scanf("%\*s"); |   连续读入单个字符且丢弃所有广义空格   |  | | --- | | scanf(" %c",...); | | |
| int printf(const char\*format,...);  成功，返回输出字符个数（'\0'不计入内）；发生错误，设置文件错误标志，返回一个负数；在写入宽字符时一个多字节的字符发生编码错误，将errno设置为EILSEQ，返回一个负数。  格式控制字符串组成：文本字符、转义序列、转换说明。  转义序列   |  |  |  |  |  | | --- | --- | --- | --- | --- | | \' |  | ' | | | | \" |  | " | | | | \? |  | ? | | | | \\ |  | \ | | | | \a | 告警或铃声 | 引发1个听得见的（铃声）或看得见的警报 | | | | \b | 退格 | 将光标 | 在当前行中向后移动1个位置 | | | \f | 新的一页或换页 | 移动到 | 新的一个逻辑页面的开始位置 | | \n | 新的一行 | 新的一行的开始位置 | | \r | 回车 | 当前行的开始位置 | | \t | 水平Tab | 下一个水平制表符位置 | | \v | 垂直Tab | 下一个垂直制表符位置 |   转换说明：%[标记][域宽][.精度][长度修饰符]转换说明符  标记   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | | | 有 | | 没有 | | | - | | | 在域宽内左对齐打印 | | 在域宽内右对齐打印 | | | + | | | 在正数前面显示1个正号 | | 不在正数前面显示1个正号 | | | 1个空格 | | | 在没有打印正号的数前面显示1个空格 | | 不在没有打印正号的数前面显示1个空格 | | | 0 | | | 在打印的数据前面加0填满域宽 | | 不在打印的数据前面加0 | | | # | 转  换  说  明  符 | o | 在输出数据的前面加上前缀0 | | 不在输出数据的前面加上前缀0 | | | x | 在输出数据的前面加上前缀0x | | 不在输出数据的前面加上前缀0x | | | X | 在输出数据的前面加上前缀0X | | 不在输出数据的前面加上前缀0X | | | e | 强制显示1个小数点 |  | 不强制显示小数点 |  | | E | | f | | g | 末尾的0不会被删除 | 末尾的0会被删除 | | G |   域宽  1、负号占用1位域宽。  2、域宽小于实际数位，向后延长。  精度   |  |  |  |  | | --- | --- | --- | --- | | 转  换  说  明  符 | 整型转换说明符 | 数据位数小于精度：  精度前无0且无小数点，数值前加前缀空格。  精度前有1个0或有1个小数点，数值前加前缀0。 | | | e、E、f | 小数点后面的数字位数。 | 数位大于精度，进行舍入处理后打印 | | g、G | 有效数字的最大位数。 | | s | 最大字符个数。 | |   域宽和精度的数可以写\*，后面实参列表中对应位置的值会替换\*。   |  |  |  | | --- | --- | --- | | 长度修饰符 | 转换说明符 | 说明 | | h短整数  l长整数 | d | 有符号10进制数 | | i | | o | 无符号8进制整数 | | u | 无符号10进制整数 | | x | 无符号16进制整数，a~f | | X | 无符号16进制整数，A~F | | L长双精度型浮点数 | e | 以指数形式显示浮点数，e | | E | 以指数形式显示浮点数，E | | f | 以小数点位置固定的形式显示浮点数 | | g | 用e（幂值小于-4或大于等于指定的精度）或f打印不带末尾0的浮点数 | | G | 用E（幂值小于-4或大于等于指定的精度）或f打印不带末尾0的浮点数 | |  | c |  | | s | 对应实参数据类型是char\*，不断打印字符直到遇到'\0' | | p | 打印1个指针 | | n | 保存当前printf语句中已经输出的字符总数，对应实参数据类型是int\*。 | | % | 显示1个百分号 | | |
| |  |  |  | | --- | --- | --- | | 数据类型 | scanf | printf | | long double长双精度浮点数 | %Lf | | | double双精度浮点数 | %lf | %f | | float浮点数 | %f | | | unsigned long int无符号长整数 | %lu | | | long int长整数 | %ld | | | unsigned int无符号整数 | %u | | | int整数 | %d | | | unsigned short无符号短整数 | %hu | | | short短整数 | %hd | | | char字符 | %c | | | |
| void perror(const char\*s);  s等于NULL或""，输出1个空格、errno对应的错误说明；s不等于NULL或""，输出1个空格、s、1个冒号、1个空格、errno对应的错误说明。 | |
| int getchar(void);  成功，返回字符的ASCII码；遇到结束符，设置文件结束标志，返回EOF；发生错误，设置文件错误标志，返回EOF。 | int putchar(int c);  成功，返回c；发生错误，设置文件错误标志，返回EOF。 |
| int fgetchar(void);  成功，返回字符的ASCII码；遇到结束符，设置文件结束标志，返回EOF；发生错误，设置文件错误标志，返回EOF。 | int fputchar(int c);  成功，返回c；发生错误，设置文件错误标志，返回EOF。 |
| char\*gets(char\*s);  成功，返回s；遇到结束符，设置文件结束标志，返回NULL（s有可能被修改）；发生错误，设置文件错误标志，返回NULL（s有可能被修改）。  '\n'被丢弃。  末尾添加'\0'。 | int puts(const char\*s);  成功，返回1个非负值；发生错误，设置文件错误标志，返回EOF。  （遇到'\0'才停止。）  追加1个'\n'。 |

操纵流的函数

杂项

|  |
| --- |
| int fileno(FILE\*stream);  成功，返回stream关联的文件的文件描述头。 |

将流与文件关联

#include<stdio.h>

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FILE\*fopen(const char\*filename,const char\*mode);  成功，返回FILE数据类型结构体的地址；失败，返回NULL。   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 文件打开模式 | 文件是否必须存在 | 是 | | | 读：  不能读  能读 | 写：  不能写  只追加写  能写 | | 否 | 存在 | 清空 | | 追加 | | 不存在，新建 | | | r(t)? | 必须存在 | |  | | 只读 | | | w(t)? |  | | 清空 | | 只写 | | | a(t)? |  | | 追加（EOF符保留） | | 追加只写 | | | r(t)?+ | 必须存在 | |  | | 读写 | | | w(t)?+ |  | | 清空 | | 读写 | | | a(t)?+ |  | | 追加（EOF符不保留） | | 读，追加写 | | | rb | 必须存在 | |  | | 只读 | | | wb |  | | 清空 | | 只写 | | | ab |  | | 追加 | | 追加只写 | | | rb+ | 必须存在 | |  | | 读写 | | | wb+ |  | | 清空 | | 读写 | | | ab+ |  | | 追加 | | 读，追加写 | | |
| FILE\*fdopen(int fildes,const char\*mode);  成功，建立1个与fildes指定的文件关联的流，返回结构体的地址。  fildes是文件描述头。  mode与fildes指定的文件的文件打开模式相符。  建立的流的文件位置指针会被设置为fildes指定的文件控制块的文件偏移量。 |
| FILE\*freopen(const char\*filename,const char\*mode,FILE\*stream);  成功，返回传入的形参stream的值；失败，返回NULL。  清除文件错误标志和文件结束标志。  filename等于NULL，修改stream的打开模式；filename指定1个文件，关闭stream连接的文件并将其与stream断开，打开filename指定的文件并与stream关联。  清除文件结束标志和文件错误标志。 |

输入输出

#include<stdio.h>（会移动文件读写位置）

程序开始执行时，自动打开stdin、stdout和stderr三个文件流。

|  |  |
| --- | --- |
| int fscanf(FILE\*stream,const char\*format,...);  成功，返回被赋值的参数的个数（可能是0）；发生错误，设置文件错误标志，返回EOF；遇到结束符，设置文件结束标志，返回EOF；在读取宽字符的时候发生编码错误，将errno设置为EILSEQ。 | int fprintf(FILE\*stream,const char\*format,...);  成功，返回输出字符个数；发生错误，设置文件错误标志，返回1个负数；在写入宽字符时一个多字节的字符发生编码错误，将errno设置为EILSEQ，返回1个负数。 |
| int getc(FILE\*stream);  成功，返回字符的ASCII码；发生错误，设置文件错误标志，返回EOF；遇到结束符，设置文件结束标志，返回EOF。  是fgetc的宏展开，可能多次计算c的值，因此表达式c不应有side-effect。 | int putc(int c,FILE\*stream);  成功，返回c；发生错误，设置文件错误标志，返回EOF。  是fputc的宏展开，可能多次计算c的值，因此表达式c不应有side-effect。 |
| int fgetc(FILE\*stream);  成功，返回字符的ASCII码；发生错误，设置文件错误标志，返回EOF；遇到结束符，设置文件结束标志，返回EOF。 | int fputc(int c,FILE\*stream);  成功，返回c；发生错误，设置文件错误标志，返回EOF。 |
| char\*fgets(char\*s,int n,FILE\*stream);  成功，返回s；遇到结束符，设置文件结束标志，返回NULL（s有可能被修改）；发生错误，设置文件错误标志，返回NULL（s有可能被修改）。  遇到'\n'或遇到'\0'或读入n-1个字符，停止。  末尾添加'\0'。 | int fputs(const char\*s,FILE\*stream);  成功，返回1个非负值；发生错误，设置文件错误标志，返回EOF。  末尾的'\0'不输出。 |
| int ungetc(int c,FILE\*stream);  成功，返回c；失败，返回EOF。  将1个字符退回到输入流中，该字符会被下一个读取输入流的函数取得。  （不会修改与流连接的文件。）  成功，取消文件结束标志，在二进制模式下，减少文件读写位置的值；在文本模式下，所有被ungetc退回的字符被读走之前文件读写位置的值是unspecified。 | |
| size\_t fread(void\*buffer,size\_t size,size\_t count,FILE\*stream);  size或count等于0，不造成任何影响，返回0；成功，返回count；发生错误，设置文件错误标志，返回不等于count的值；遇到结束符，设置文件结束标志，返回不等于count的值。  buffer是内存区域的起始地址，例如：&**变量名**、**数组名**。  size是每个元素占的字节数。  count是元素个数。 | size\_t fwrite(const void\*buffer,size\_t size,size\_t count,FILE\*stream);  size或count等于0，不造成任何影响，返回0；成功，返回count；发生错误，设置文件错误标志，返回不等于count的值。  buffer是内存区域的起始地址，例如：&**变量名**、**数组名**。  size是每个元素占的字节数。  count是元素个数。 |

#include<conio.h>

|  |  |
| --- | --- |
| int getch(void);  成功，返回字符的ASCII码。  从控制台读取1个字符，但不把该字符显示在屏幕上。  不等待用户按回车就立刻返回。  按Enter键返回的字符为'\r'。不能读取ctrl+C。按Fn或箭头时，第1次调用返回0或0xE0，第2次调用返回实际的键值。 | int putch(int c);  成功，返回字符的ASCII码；发生错误，返回EOF。 |
| int getche(void);  成功，返回字符的ASCII码。  从控制台读取1个字符，该字符会显示在屏幕上。  不等待用户按回车就立刻返回。  按Enter键返回的字符为'\r'。 | |

关闭流

#include<stdio.h>

|  |
| --- |
| int fclose(FILE\*stream);  成功，返回0；失败，返回EOF。  与stream关联的所有输入、输出缓冲区中的内容都被丢弃。  即使失败，stream也与文件及其所有缓冲区断开。 |

文件读写位置

|  |
| --- |
| long ftell(FILE\*stream);  成功，返回文件读写位置；失败，返回-1。  在文本模式下，返回值可能无意义但可以被fseek使用。  在文本模式下，若有ungetc退回的字符没被读走，未定义。 |
| int fseek(FILE\*stream,long offset,int whence);  成功，返回0；发生错误，设置文件错误标志，返回非0值。  清除文件结束标志。  发生错误，文件读写位置不变。  ungetc退回的字符会被丢弃。  在文本模式下，offset等于0或ftell的返回值，whence等于SEEK\_SET。  whence   |  |  |  | | --- | --- | --- | |  | Binary files | Text files | | SEEK\_SET | 从文件开头移动 | | | SEEK\_CUR | 从文件读写位置移动 |  | | SEEK\_END | 从文件结尾移动 |  | |
| void rewind(FILE\*stream);  清除文件错误标志和文件结束标志。  ungetc退回的字符会被丢弃。 |
| int fgetpos(FILE\*stream,fpos\_t\*pos);  成功，返回0；发生错误，返回非0值。  应在调用之前为pos分配内存空间。 |
| int fsetpos(FILE\*stream,const fpos\_t\*pos);  成功，返回0；发生错误，返回非0值。  清除文件结束标志。  ungetc退回的字符会被丢弃。 |

文件错误标志和文件结束标志（调用clearerr,rewind,fseek,fsetpos或freopen可清除其中至少1个）

#include<stdio.h>

|  |
| --- |
| int feof(FILE\*stream);  文件结束标志被设置，返回非0值；否则返回0。 |
| int ferror(FILE\*stream);  文件错误标志被设置，返回非0值；否则返回0。 |
| void clearerr(FILE\*stream);  清除stream的文件错误标志和文件结束标志（使文件错误标志和文件结束标志置为0）。 |

缓冲区分类

<http://www.chinaunix.net/old_jh/23/588099.html>

<http://outofmemory.cn/C-lang/tutorial/4-FILE-structure-buffer-understand>

|  |  |  |
| --- | --- | --- |
| 全缓冲 | 当填满标准I/O缓存后才进行实际I/O操作。 | 对磁盘文件的读写 |
| 行缓冲 | 当在输入和输出中遇到换行符时，执行真正的I/O操作。 | stdin、stdout |
| 不带缓冲 | 不进行缓冲。 | stderr |

缓冲区的刷新：下一个要被读取的字符的地址(ptr)设置为缓冲区的基地址(base) ，同时剩余的字符个数(cnt)的值设置为0。

操纵缓冲区的函数

|  |
| --- |
| void setbuf(FILE\*stream,char\*buffer);  buffer不等于NULL，相当于setvbuf(stream,buffer,\_IOFBF,BUFSIZ)；buffer等于NULL，相当于setvbuf(stream,NULL,\_IONBF,1个会被忽略的整数)。  在打开文件之后、对文件进行所有输入、输出操作之前，才能调用。  若buffer不等于NULL，buffer指向的内存空间大小至少占BUFSIZ个字节。 |
| void setlinebuf(FILE\*stream);  相当于setvbuf(stream, NULL,\_IOLBF,0)。  若之前有缓冲区，缓冲区内数据被丢弃。 |
| int setvbuf(FILE\*stream,char\*buffer,int mode,size\_t size);  成功，返回0；否则返回非0值。  在打开文件后，对文件进行所有输入、输出操作之前，才能调用。  缓存类型   |  |  |  | | --- | --- | --- | |  | input | output | | \_IOFBF  全缓存 | 要求进行输入操作且缓存区为空时，填充缓冲区。 | 缓冲区填满或执行flush操作时，输出。 | | \_IOLBF  行缓存 | 要求进行输入操作且缓存区为空时，填充缓冲区直到遇到下一个'\n'。 | 缓冲区填满或输入'\n'或执行flush操作时，输出。 | | \_IONBF  不带缓存 | 直接输入和输出。  忽略buffer和size。 | |   若buffer不等于NULL，buffer指向的size个字节的内存空间被用作缓冲区；若buffer等于NULL，给buffer分配size个字节的内存空间并用作缓冲区，流被关闭后buffer指向的内存空间被释放。 |
| int fflush(FILE\*stream);  成功或stream无缓冲区或stream是只读打开的，返回0；发生错误，设置文件错误标志，返回EOF。  若stream指向输出流或最近执行的操作不是输入的更新流，把缓冲区中所有待写数据传送至宿主环境（host environment）写入文件；若stream等于NULL，所有符合条件的流都受到作用；否则（例如：stdin），未定义（一些编译器的实现方法是清空缓冲区）。 |

获取文件（包括目录）的函数

#include<dirent.h><https://github.com/tronkko/dirent>

|  |
| --- |
| DIR\*opendir(const char\*dirname);  成功，返回DIR\*形态的目录流；失败，返回NULL。  目录流的读写位置被设置在第1个目录项。 |
| dirent\*readdir(DIR\*dirp);  成功，返回位于目录流的读写位置的目录项；目录流的读写位置位于目录结尾或失败，返回NULL。  会移动目录流的读写位置。 |
| int closedir(DIR\*dirp);  成功，返回0；失败，返回-1。 |
| void rewinddir(DIR \*dirp);  将目录流的读写位置设置在第1个目录项。 |

#include<tinydir.h><https://github.com/cxong/tinydir>

|  |
| --- |
|  |

#include<Windows.h>

|  |
| --- |
| HANDLE FindFirstFile(LPCTSTR lpFileName,LPWIN32\_FIND\_DATA lpFindFileData);  成功，返回1个搜索句柄；发生错误（例如：没有找到文件），返回INVALID\_HANDLE\_VALUE。  成功，lpFindFileData存储第1个找到的目录或文件的信息；出现错误，lpFindFileData内容不确定。 |
| BOOL FindNextFile(HANDLE hFindFile,LPWIN32\_FIND\_DATA lpFindFileData);  成功，返回非0值；发生错误（例如：没有找到更多文件），返回0。  成功，lpFindFileData存储下1个找到的目录或文件的信息；出现错误，lpFindFileData内容不确定。 |
| BOOL FindClose(HANDLE hFindFile);  成功，返回非0值；发生错误，返回0。 |

操纵目录的函数

#include<Windows.h><https://msdn.microsoft.com/zh-cn/library/windows/desktop/aa363950(v=vs.85).aspx>

|  |
| --- |
| BOOL CreateDirectory(LPCTSTR lpPathName,LPSECURITY\_ATTRIBUTES lpSecurityAttributes);  成功，返回非0值；发生错误，返回0。 |
| BOOL SetCurrentDirectory(LPCTSTR lpPathName);  成功，返回非0值；发生错误，返回0。 |
| BOOL RemoveDirectory(LPCTSTR lpPathName);  成功，返回非0值；发生错误，返回0。  lpPathName指定的目录是空的。 |
| BOOL MoveFile(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName);  成功，返回非0值；发生错误，返回0。  lpNewFileName指定的目录不存在。  lpExistingFileName指定的目录和lpNewFileName指定的目录在同一个驱动器上。  目录的子项也被移动。 |

#include<stdlib.h><http://www.computerhope.com/msdos.htm>

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| int system(const char\*command);  command   |  |  | | --- | --- | | 创建目录  md "目录路径"  mkdir "目录路径" | 1、若目录路径不是绝对路径，则从当前目录定位目录。  2、当仅当目录路径不含空格时，双引号可以省略。  3、若路径上有不存在的目录，则也都创建。 | | 移动[当前目录位置]  cd "目录路径"  chdir "目录路径" | 1、若目录路径不是绝对路径，则从当前目录定位目录。  2、修改shell的[当前目录位置]，不修改进程的[当前目录位置]。 | | 删除空目录  rd "目录路径"  rmdir "目录路径" | 1、若目录路径不是绝对路径，则从当前目录定位目录。  2、当仅当目录路径不含空格时，双引号可以省略。 | | 重命名目录  ren "目录路径" "目录名"  rename "目录路径" "目录名" | 1、若目录路径不是绝对路径，则从当前目录定位目录。  2、当仅当目录路径不含空格时，双引号可以省略。  3、当仅当目录名不含空格时，双引号可以省略。 | |

复制目录和删除目录

<http://www.cnblogs.com/xianyunhe/archive/2011/12/06/2278550.html>

操纵文件的函数

#include<Windows.h><https://msdn.microsoft.com/zh-cn/library/windows/desktop/aa364232(v=vs.85).aspx>

|  |
| --- |
| BOOL CopyFile(LPCTSTR lpExistingFileName,LPCTSTR lpNewFileName,BOOL bFailIfExists);  成功，返回非0值；发生错误，返回0。  bFailIfExists等于TRUE，lpNewFileName指定的文件存在，失败；bFailIfExists等于FALSE，lpNewFileName指定的文件存在，覆盖该文件，成功。 |
| BOOL DeleteFile(LPCTSTR lpFileName);  成功，返回非0值；发生错误，返回0。 |
| BOOL MoveFile(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName);  成功，返回非0值；发生错误，返回0。  lpNewFileName指定的文件不存在。 |

#include<stdlib.h>

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| int system(const char\*command);  command   |  |  | | --- | --- | | 复制文件  copy "文件路径" "目录路径" | 1、若文件路径不是绝对路径，则从当前目录定位文件。  2、若目录路径不是绝对路径，则从当前目录定位目录。  3、当仅当文件路径不含空格时，双引号可以省略。  4、当仅当目录路径不含空格时，双引号可以省略。 | | 删除文件  erase "文件路径"  del "文件路径" | 1、若文件路径不是绝对路径，则从当前目录定位文件。  2、当仅当文件路径不含空格时，双引号可以省略。 | | 移动文件  move "文件路径" "目录路径" | 1、若文件路径不是绝对路径，则从当前目录定位文件。  2、若目录路径不是绝对路径，则从当前目录定位目录。  3、当仅当文件路径不含空格时，双引号可以省略。  4、当仅当目录路径不含空格时，双引号可以省略。 | | 重命名文件  ren "文件路径" "文件名"  rename "文件路径" "文件名" | 1、若文件路径不是绝对路径，则从当前目录定位文件。  2、当仅当文件路径不含空格时，双引号可以省略。  3、当仅当文件名不含空格时，双引号可以省略。 | |

【C++】CString, BSTR, LPCTSTR之间的关系和区别 <http://www.cnblogs.com/dracohan/archive/2009/02/25/1397763.html>

Windows Data Types <https://msdn.microsoft.com/en-us/library/windows/desktop/aa383751(v=vs.85).aspx>

dll

<http://m.blog.csdn.net/article/details?id=48929865>

<http://www.jellythink.com/archives/111>

<https://msdn.microsoft.com/en-us/library/ms235636.aspx>

<http://c.biancheng.net/cpp/html/2753.html> <http://c.biancheng.net/cpp/html/2754.html>

Dynamic-Link Libraries <https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx>

Dynamic-Link Library Functions <https://msdn.microsoft.com/en-us/library/windows/desktop/ms682599(v=vs.85).aspx>

Process and Thread Functions <https://msdn.microsoft.com/zh-cn/ms684847>

DLLs in Visual C++ <https://msdn.microsoft.com/en-us/library/1ez7dh12.aspx>

线程局部存储<http://www.cnblogs.com/stli/archive/2010/11/03/1867852.html>