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/>

<https://stackoverflow.com/questions/81656/where-do-i-find-the-current-c-or-c-standard-documents>

|  |  |
| --- | --- |
| 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| c89(withdrawn) | 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| c90(withdrawn) | 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| c99(withdrawn) | 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。 |

\回车会被忽略（不管在哪里出现，都会被忽略）。

# 运算符

1、表达式的求值过程可能有副作用，例如：变量的值被修改、文件被打开、文件的内容被修改、文件被关闭。因此是否去求表达式的值以及求多少次也都需要被关注。若表达式不可能产生副作用且值不被结构化语句使用，则不求值。

|  |  |
| --- | --- |
| struct c  {  int a;  };  int main(void)  {  struct c\*p;  p->a;  } | struct c  {  int a;  };  int main(void)  {  struct c\*p=(struct c\*)123;  p->a;  } |

2、优先级决定两个拥有共同操作数且优先级不同的运算符的执行顺序，结合顺序决定两个拥有共同操作数且优先级相同的运算符的执行顺序。运算符的操作数的求值顺序决定一些不相邻的表达式的求值顺序。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 优先级 |  | 用法 | 计算结果 | 副作用 |  | 结合顺序 |
|  | ()括号 |  | 改变运算符的结合顺序 |  |  |  |
|  |  |  | 编译器自动连接相邻的字符串 |  | 1、转义字符在连接字符串前转义。   |  | | --- | | #include<stdio.h>  int main(void)  {  printf("%s\n","999 \1""01 999");  printf("%s\n","999 \101 999");  } | |  |
| 1 | ()函数调用 | **表达式1**((**表达式2**(,**表达式**)\*)?) | 返回值数据类型不为void的函数返回1个值。 | 调用的函数产生的副作用。 | ！！！这里的逗号是分隔符，不是逗号运算符。  1、表达式1是函数，例如：函数名。 | → |
| []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**)。因此表达式1和表达式2可以交换位置。  2、表达式1是数组的1个元素的地址，例如：数组名（数组名参与sizeof和一元&以外的运算时自动转换为指向首元素的指针）。  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。 |
| .**成员名**通过对象选择成员 | **表达式**.**成员名** | 得到对象的1个成员。 |  |  |
| ->**成员名**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。 | 使用表达式的值后使表达式的值加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。 | 使用表达式的值后使表达式的值减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。 |
| 2 | ++一元前置递增 | ++**表达式** | 得到表达式的值。 | 表达式求值之前使表达式的值加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。 | ← |
| --一元前置递减 | --**表达式** | 得到表达式的值。 | 表达式求值之前使表达式的值减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。 |
| (**数据类型名**)C风格的一元强制数据类型转换（指派） | (**数据类型名**)**表达式** | 为操作数创建1个目标数据类型的临时副本。 |  | ！！！得到的值是右值。  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（间接运算符、取值运算符） | \***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（指针运算符、地址运算符） | &**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** | 得到数据类型或表达式的值占的字节数，数据类型是size\_t（某无符号整型的别名）。 |  | ！！！不求表达式的值。  ！！！sizeof **表达式**是常量表达式。  1、C把char数据类型的长度定义为1字节。  2、sizeof(**数组名**)得到数组占的字节数，sizeof(**数组名**+0)得到指针数据类型占的字节数（4），因为数组名参与sizeof和一元&以外的运算时自动转换为指向首元素的指针。  3、sizeof(void)、sizeof((void)**表达式**)、sizeof(**调用返回值数据类型为void的函数**)得到0。 |
| 3 | \*乘 | **表达式**\***表达式** |  |  |  | → |
| /除以 | **表达式1**/**表达式2** |  |  | 1、表达式2的值不等于0。  2、整数除法截尾取整。 |
| %求模 | **表达式1**%**表达式2** |  |  | 1、表达式1和表达式2的数据类型是整型。  2、表达式2的值不等于0。  3、整数除法余数符号与被除数相同。a-a/b\*b和a%b相等。 |
| 4 | +加 | **表达式**+**表达式** |  |  |  | → |
| -减 | **表达式**-**表达式** |  |  |  |
| 5 | <<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、表达式2的数据类型是整型。  2、右边腾空的数位补0。 | → |
| >>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、表达式2的数据类型是整型。  2、左边腾空的数位的填补方式取决于计算机。 |
| 6 | <小于 | **表达式**<**表达式** | 成立，得到1；不成立，得到0。 |  |  | → |
| <=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** |  |  |
| 7 | ==关系等于 | **表达式**==**表达式** |  |  | → |
| !=关系不等于 | **表达式**!=**表达式** |  |  |
| 8 | &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。 | → |
| 9 | ^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。 | → |
| 10 | |按位或 | **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。 | → |
| 11 | &&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。 | → |
| 12 | ||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。 | → |
| 13 | ?: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。 |  | ！！！得到的值是右值。  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。 | ← |
| 14 | =赋值 | **表达式1**=**表达式2** | 得到赋值结束后表达式1的值。 | 将表达式2的值赋给表达式1。 | ！！！得到的值是右值。  1、表达式1的值是左值，访问限定没有设置const。 | ← |
| +=加赋值 | **表达式1**+=**表达式2** | （略） | （略） | 1、相当于**表达式1**=**表达式1** **运算符**(**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** |
| %=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** |
| 15 | ,逗号 | **表达式1**,**表达式2** | 得到表达式2的值。 |  | ！！！得到的值是右值。  1、先求表达式1的值，后求表达式2的值。 | → |

运算符的操作数的求值顺序（定值顺序）

|  |  |
| --- | --- |
| 逻辑与&&、逻辑或||、逗号, | 从左向右 |
| 其它二元运算符、函数调用() | 未定义 |
| 条件运算符?: | 先左，后（左不等于0）中或（左等于0）右 |

顺序点（序列点、序列的分界点）：程序执行中的一点。在顺序点处，所有副作用在继续进行之前生效。

1、完整表达式（不是1个更大的表达式的子表达式的表达式）的结束是1个顺序点。

2、逻辑与&&是1个顺序点。逻辑或||是1个顺序点。逗号,是1个顺序点。

3、分号;是1个顺序点。

4、若1个左值在1个表达式中多次出现且每处参与运算的先后顺序不能根据规则确定（未定义），则不对它使用有副作用的运算符。

|  |
| --- |
| #include<stdio.h>  int main(void)  {  int a=1;  printf("%d\n",a);  printf("%d %d %d %d %d %d\n",++a,a,++a,a,++a,a);错误，运行结果依赖于未定义的行为  printf("%d\n",a);  printf("%d %d %d %d %d %d\n",a++,a,a++,a,a++,a);错误，运行结果依赖于未定义的行为  } |

#include<iso646.h>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 |  |  | not | ! |
| compl | ~ |  |  |
| 7 |  |  | not\_eq | != |
| 8 | bitand | & |  |  |
| 9 | xor | ^ |  |  |
| 10 | bitor | | |  |  |
| 11 |  |  | and | && |
| 12 |  |  | or | || |
| 14 | and\_eq | &= |  |  |
| xor\_eq | ^= |  |  |
| or\_eq | |= |  |  |

# 标识符、关键字

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 标识符作用域   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 函数作用域 | 标签（标号） | 在函数内定义 | 在函数中被访问 |  | | 文件作用域 | ... | 在所有函数外定义 | 从声明到当前文件结束被访问 | 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**能直接被访问的区域**外标识符**不能直接被访问。  3、1条结构化语句相当于包含它的所有语句的1个代码块。   |  | | --- | | int main(void)  {  int i=1;  for(int i=0;i!=1;++i)  {  int i=-1;  }  } | | | 代码块作用域 | ... | 在代码块内定义 | 从声明到当前代码块结束被访问 | | 函数原型作用域 | 函数原型的形参名 | 在函数原型内定义 | 从声明到函数原型结束被访问 |  | |
| 标识符链接   |  |  | | --- | --- | | 空链接 | 由定义所在的代码块或函数原型私有 | | 内部链接 | 由定义所在的文件私有 | | 外部链接 | 可以被其它文件使用（需要引用声明） | |

变量存储时期（存储周期）

|  |  |
| --- | --- |
| 静态存储时期 | 程序执行期间存在 |
| 自动存储时期 | 代码块执行期间存在 |
| 分配的（还不一定有标识符） | 从调用内存分配函数到调用内存释放函数/从调用内存分配函数到程序结束 |

变量存储类型

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 定义声明 | | 引用声明 | |
| 自动 | 空链接 | 自动存储时期 | 代码块内，(auto)? | 从声明到当前代码块结束 | / | / |
| 寄存器 | 代码块内，register | / | / |
| 空链接的静态 | 静态存储时期 | 代码块内，static | / | / |
| 具有内部链接的静态 | 内部链接 | 所有函数外，static | 从声明到当前文件结束 | 代码块内或所有函数外，extern | 从声明到当前代码块结束/从声明到当前文件结束 |
| 具有外部链接的静态 | 外部链接 | 所有函数外，无关键字 | 代码块内或所有函数外，extern |
| typedef |  |  |  | 从声明到当前代码块结束/从声明到当前文件结束 |  |  |

|  |  |  |
| --- | --- | --- |
| 自动 | 1、若没有初始化，赋值之前变量的值未定义。 | 变量存储在内存中。  1、函数头中的形参和代码块中定义的变量的存储类型默认是auto。  2、编译器可能将auto存储类型变为register存储类型。不省略auto表示不能把变量改变为其它存储类型。 |
| 寄存器 | 变量存储在寄存器中。  1、不能取变量的地址。  2、编译器可能忽略register（当然，忽略了也不能取变量的地址）。 |
| 空链接的静态 | 1、没有初始化相当于初始化为0。  2、至多初始化1次。 | 1、在代码块内定义的静态变量可以在代码块外间接访问。   |  | | --- | | int\*f(void)  {  static int a;  return&a;  }  int main(void)  {  int\*a=f();  \*a=1;  } | |
| 具有内部链接的静态 | 1、可以用引用声明在当前文件内扩展标识符的作用域。   |  | | --- | | extern int n;  int main(void)  {  n=1;  }  static int n; | |
| 具有外部链接的静态 | 1、可以用引用声明扩展标识符的作用域。   |  |  |  | | --- | --- | --- | | int a; | void f()  {  extern int a;  a=1;  } | extern int a;  int main(void)  {  a=1;  } | |
| typedef |  | 创建数据类型的别名  1、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。 |

变量访问限制（访问限定符（类型限定词）可以使用任意多个）

|  |  |
| --- | --- |
| const | 不能通过该标识符修改数据。（仍然可能可以通过其他途径修改数据。） |
| volatile | 限制编译器的优化。  变量除了可被当前程序改变还可以被其它程序改变。因此程序每次读取这个变量的值的时候，都会直接从变量地址中读取数据（而不是使用寄存器中可能是过期的值）。 |
| restrict | 限制编译器的优化。  只适用于指针。该指针是访问所指数据块的唯一且初始的方式。即任意时刻所有restrict指针所指数据块互不重叠。  <https://www.zhihu.com/question/41653775> |

函数存储类型

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | 定义声明 | | 引用声明 | |
| 静态的 | 内部链接 | 所有函数外，static | 从声明到当前文件结束 | 代码块内或所有函数外，static | 从声明到当前代码块结束/从声明到当前文件结束 |
| 外部的 | 外部链接 | 所有函数外，无关键字 | 代码块内或所有函数外，(extern)? |
| 内联 | 内部链接 | 所有函数外，inline | 代码块内或所有函数外，inline |

|  |  |
| --- | --- |
| 静态的 |  |
| 外部的 |  |
| 内联 | 1、如果对内联函数取地址，编译器将它转为外部的。 |

！！！如果可以同时访问内部定义和外部定义，编译器可以随意使用内部定义和外部定义。

标识符

1、标识符由小写字母、大写字母、数字、下划线组成，第1个字符是小写字母、大写字母、或下划线。

2、标识符区分大小写。

3、编译器只识别局部标识符的前63个字符和外部标识符的前31个字符。

|  |  |  |
| --- | --- | --- |
|  | 定义声明（存储类型修饰符auto、register、static、extern、typedef至多使用1个）  至多出现1次。若标识符被使用，恰好出现1次。 | 引用声明 |
| 数据类型 | 结构体定义声明 | 结构体引用声明 |
| 共用体定义声明 | 共用体引用声明 |
| 枚举定义声明 | 枚举引用声明 |
| 函数 | 函数定义声明 | 函数引用声明 |
| 若在声明前使用，则假定返回值数据类型是int。   |  |  |  |  | | --- | --- | --- | --- | | int main(void)  {  f(1);  f(1,1);  f(1,1,1);  }  int f(int a)  {  return 0;  } | int main(void)  {  f(1);  f(1,1);  f(1,1,1);  }  short f(int a)  {  return 0;  } | int main(void)  {  f(1);  f(1,1);  f(1,1,1);  }  long f(int a)  {  return 0;  } | int main(void)  {  f(1);  f(1,1);  f(1,1,1);  }  double f(int a)  {  return 0;  } | | 正确 |  |  | 错误 | | |
| 变量（不包括函数） | 确定变量的存储类型，为变量分配存储空间。 | 向编译器描述变量的数据类型。  1、不初始化。 |

保留标识符：printf等。

关键字

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| auto  register  static  inline  typedef | extern | const  volatile  restrict | |  | | --- | | void | | signed  unsigned  char  short  int  long  float  double | | struct  union  enum | | \_Bool  \_Complex  \_Imaginary | | |  | | --- | | if  else  switch  case  default | | while  for  do | | break  continue  return  goto | | sizeof |

# 数据

## 基本数据类型

|  |  |  |
| --- | --- | --- |
|  | ANSI C规定的最小值 | 文字（非符号常量） |
| void | / |  |
| \_Bool（某无符号整数的别名）  #include<stdbool.h>   |  |  | | --- | --- | | bool | \_Bool | | / | 0、1  #include<stdbool.h>   |  |  | | --- | --- | | false | 0 | | true | 1 | |
| signed char  unsigned char | 8位 | 可以用转义字符表示不可打印字符。   |  |  |  |  |  | | --- | --- | --- | --- | --- | | \' |  | '  在字符串常量中可以直接写'。 | | | | \" |  | "  在字符常量中可以直接写"。 | | | | \? |  | ? | | | | \\ |  | \ | | | | \a警报字符 | 告警或铃声 | 引发1个听得见的（铃声）或看得见的警报 | | | | \b退格符 | 退格 | 将光标 | 在当前行中向后移动1个位置 | | | \f走纸符 | 新的一页或换页 | 移动到 | 新的一个逻辑页面的开始位置 | | \n换行符 | 新的一行 | 新的一行的开始位置 | | \r回车符 | 回车 | 当前行的开始位置 | | \t水平制表符 | 水平Tab | 下一个水平制表符位置 | | \v垂直制表符 | 垂直Tab | 下一个垂直制表符位置 | | \**o**  \**oo**  \**ooo** |  | **o**表示1位八进制数字。  \**o**和\**oo**的格式需要接下来的字符不是0~7。   |  | | --- | | #include<stdio.h>  int main()  {  printf("%s\n%s\n%s",  "\0616",输出16  "\616",错误，数太大了  "\618");输出18  return 0;  } | | | | | \x(**h**)\* |  | **h**表示1位十六进制数字。  需要接下来的字符不是0~9或A~F或a~f。 | | |   \t和制表符等价。  单引号内有多个字符的字符常量的值是每个字符的值按顺序拼接的结果。例：   |  | | --- | | #include<stdio.h>  void \_print32(unsigned int a)  {  unsigned int i,b=1<<31;  for(i=0;i!=32;++i)  {  if(i%8==0)printf(" ");  printf("%d",(a&b)==b);  b>>=1;  }  printf("\n");  }  int main(void)  {  unsigned int c='abcd';  \_print32(c);  } | |
| #include<stdlib.h>/#include<wchar.h>   |  |  | | --- | --- | | wchar\_t | 整数类型，表示支持的场所指定的最大扩展字符集 | |  |  |
| #include<wchar.h>/#include<wctype.h>   |  |  | | --- | --- | | wint\_t | 整数类型，可以保存扩展字符集的任何值和至少一个不是扩展字符集成员的值 | |  | #include<wchar.h>/#include<wctype.h>   |  |  | | --- | --- | | WEOF | 表示宽字符输入的文件结尾 | |
| (signed )?short( int)?  unsigned short( int)? | 16位 | 无 |
| signed( int)?|int  unsigned( int)? | 16位 | |  |  | | --- | --- | |  | 前缀 | | 八进制 | 0 | | 十进制 | 无 | | 十六进制 | 0x或0X |   十进制默认顺序：int、long、unsigned long、long long/unsigned long long。  （？？？无法以十进制书写unsigned int字面值常量）  八进制和十六进制默认顺序：int、unsigned int、long、unsigned long、long long/unsigned long long。   |  |  | | --- | --- | |  | 后缀 | | 强制转换为unsigned int | u或U | | 强制转换为long | l或L | | 强制转换为unsigned long | ul或UL或lu或LU | | 强制转换为long long | ll或LL | | 强制转换为unsigned long long | ull或ULL或llu或LLU | |
| (signed )?long( int)?  unsigned long( int)? | 32位 |
| (signed )?long long( int)?  unsigned long long( int)? | 64位 |
| float | 6位有效数字 | 字面值浮点数常量的数据类型默认是double。   |  |  |  | | --- | --- | --- | |  | 前缀 | 指数计数法 | | 十进制 | 无 | 用e或E分隔尾数和指数。 | | 十六进制 | 0x或0X | 用p或P分隔尾数和指数，尾数十六进制，阶码十进制，阶码是2的指数。 |  |  |  | | --- | --- | |  | 后缀 | | 强制转换为float | f或F | | 强制转换为long double | l或L |   特殊浮点数  #include<math.h>   |  |  | | --- | --- | | NAN | 当且仅当实现支持float的NaN时被定义。 | | INFINITY | 如果有，展开为正的或无符号的无穷大的常量，数据类型是float。如果没有，展开为一个在编译时溢出的正的浮点数常量。 |   <https://zh.wikipedia.org/wiki/%E6%93%B4%E5%B1%95%E5%AF%A6%E6%95%B8%E7%B7%9A>   |  |  | | --- | --- | | 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、负无穷加负无穷、正无穷乘负数、负无穷乘正数 | | 1.#QNAN(0)\*  -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 | |
| double | 10位有效数字 |
| long double | 10位有效数字 |
| float \_Complex  double \_Complex  long double \_Complex  float \_Imaginary  double \_Imaginary  long double \_Imaginary  #include<complex.h>   |  |  | | --- | --- | | complex | \_Complex | | imaginary | \_Imaginary | |  | #include<complex.h>   |  |  | | --- | --- | | \_Complex\_I | const float \_Complex数据类型的表达式 | | \_Imaginary\_I | const float \_Imaginary数据类型的表达式 | | I | \_Complex\_I或\_Imaginary\_I | |

void用于表示函数不接收参数、函数不返回值。指针的数据类型是void\*表示提供指针的人没有或不需要确定指针指向什么数据类型。

char是signed char的别名或unsigned char的别名，由编译器决定。

C把char数据类型的长度定义为1字节。

<https://www.zhihu.com/question/20084419>

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 |

|  |  |  |
| --- | --- | --- |
| 变量的定义声明、引用声明 | **基本数据类型名**( **访问限定符**)\* **变量名**(=**表达式**)?(,  (**访问限定符** )\***变量名**(=**表达式**)?)\*;  根据存储类型，在前面加上存储类型修饰符。 | 1、不能声明void数据类型的变量。  2、若是定义声明且访问限定符列表中有const，则必须初始化。  3、变量的数据类型是**基本数据类型**( **访问限定符**)\*。  4、**基本数据类型名 访问限定符**和**访问限定符 基本数据类型名**等价。 |

#include<stdlib.h>/#include<string.h>/#include<wchar.h>

|  |  |
| --- | --- |
| size\_t | sizeof运算符返回的整数类型 |

## 自定义数据类型

结构体

|  |  |  |  |
| --- | --- | --- | --- |
| 定义声明 | struct( **结构体标签**)?  {  ( **变量声明语句**  )+}; | 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 **结构体标签**)和sizeof **表达式**；若在代码块内声明，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。）   |  | | --- | | struct **结构体标签 成员名**[1][1][1][1];错误  struct **结构体标签**(\***成员名**[1][1])[1][1];正确 |   3、0长度数组成员<http://coolshell.cn/articles/11377.html>。  4、声明可以嵌套。 |
| 引用声明 | struct **结构体标签**; |  |
| 变量的定义声明、引用声明 | struct **结构体标签** (**访问限定符** )\***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、若数据类型在代码块内声明且包含访问限定设置const的成员，则必须初始化。 |
| 操作 | =赋值  &取地址  .**成员名**通过对象选择成员  sizeof按字节确定大小 | 1、赋值操作也会复制数组成员的每一个元素。  ！！！不能用==、!=比较两个结构体。  ！！！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。 |
| 在定义声明数据类型时声明变量 | 在变量声明语句中将struct **结构体标签**换成**[定义声明]去掉末尾的分号**。 |  |
| 在定义声明数据类型中使用typedef | 对[在定义声明数据类型时声明变量]使用typedef。 |  |

共用体

|  |  |  |
| --- | --- | --- |
| 定义声明 | union( **数据类型名**)?  {  ( **变量声明语句**  )+}; |  |
| 引用声明 | 与结构体类似。 |  |
| 变量的定义声明、引用声明 | union **数据类型名** (**访问限定符** )\***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。 |
| 操作 | =赋值  &取地址  .**成员名**通过对象选择成员  sizeof按字节确定大小 | ！！！不能用==、!=比较两个共用体。 |
| 在定义声明数据类型时声明变量 | 与结构体类似。 |  |
| 在定义声明数据类型中使用typedef | 对[在定义声明数据类型时声明变量]使用typedef。 |  |

枚举（数据）类型

|  |  |  |
| --- | --- | --- |
| 定义声明 | enum **数据类型名**{(**标识符**(=**表达式**)?(,**标识符**(=**表达式**)?)\*)?}; | 1、表达式是常量表达式，数据类型是整型。  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。 |
| 引用声明 | 与结构体类似。 |  |
| 变量的定义声明、引用声明 | enum **数据类型名** (**访问限定符** )\***变量名**(=(**标识符**|(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。 |
| 在定义声明数据类型时声明变量 | 与结构体类似。 |  |
| 在定义声明数据类型中使用typedef | 对[在定义声明数据类型时声明变量]使用typedef。 |  |

## （某数据类型的）派生数据类型

（**某数据类型**的）指针

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 定义声明、引用声明 | 在**某数据类型**的变量声明语句中，   |  |  | | --- | --- | | 将 | 换为 | | **变量名** | \*(const )?**变量名** |   若变量名后紧接着[，则需要加1层括号将\*和变量名括起来（因为[]优先级更高）。  可以声明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（最左侧的基本数据类型当作1个\*处理）插入1个\*。若插入的\*后紧接着[，则必须加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、初始化、传参、返回值的规则都与赋值的规则相同。  3、若是定义声明且最后一个\*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。）   |  |  | | --- | --- | | int const \*const \* \*const p=NULL;  p=NULL;错误  \*p=NULL;正确  \*\*p=NULL;错误  \*\*\*p=0;错误 | #include<stdio.h>  void f(int const\*p1,int\*p2)  {  printf("%d %d\n",\*p1,\*p2);  \*p2=1;  printf("%d %d\n",\*p1,\*p2);  }  int main(void)  {  int a=0;  f(&a,&a);  } | |
| 操作 | =赋值  \*解引用  &取地址  sizeof按字节确定大小  ->**成员名**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** | ！！！语言不负责内存边界检查。  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、表达式的数据类型是整型。  3、**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。（若1个\*没有const，其左侧的所有\*都没有const。）（最左侧的基本数据类型也当作1个\*处理。）<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;  允许的情况有且只有：  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;  int const \*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;  int const \*const \*const \*const \*const \*const \*const \*const c7=&a; | | int \* \*const \* \*const \* \* a=0;  允许的情况有且只有：  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;  int const \*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;  int const \*const \*const \*const \*const \*const \*const \*const c6=&a; | |
| 1、指针+整数和整数+指针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**)。两个指针指向同一数组的元素时才有意义。  3、取地址**新数据类型**的表达式=&**某数据类型**的表达式;。  4、**新数据类型**的表达式->成员名等价于(\***新数据类型**的表达式).成员名。  5、\*p+=1等价于(\*p)++。 |

（**某数据类型**的）数组

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 定义声明 | 在**某数据类型**的变量声明语句中，   |  |  | | --- | --- | | 将 | 换为 | | **变量名** | **变量名**[**元素个数**] | | (=**表达式**)? | (={([**下标**]=)?**表达式**(,  ([**下标**]=)?**表达式**)\*})? |   或   |  |  | | --- | --- | | 将 | 换为 | | **变量名** | **变量名**[] | | (=**表达式**)? | ={**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（最左侧的基本数据类型当作1个\*处理）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、若最后一个\*后（最左侧的基本数据类型当作1个\*处理）有const，则必须初始化。  3、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。  4、第1个表达式不指定下标相当于指定为0。除了第1个表达式，没有指定下标的表达式初始化的元素的下标等于前1个的下标加1。若两个表达式对同一个元素初始化，后面的覆盖前面的。  5、**元素个数**是常量表达式。   |  | | --- | | 若编译器支持变长数组，可以使用变长数组数据类型，用变量定义数组每维长度。  变长数组的存储类型是auto。  变长数组定义声明时不初始化。 | |
| 1、**某数据类型**的变量也可以用这种方式初始化，初始化表达式个数为1。  2、char s[]={('**字符**',)\*'\0'};等价于char s[]="(**字符**)\*";。 |
| 引用声明 |  | 1、既不指定元素个数也不提供初始化值。 |
| 操作 | sizeof按字节确定大小  &取地址  ...（对其它运算符，数组名自动转换为1个值不能被修改的指向首元素的指针。<https://zhuanlan.zhihu.com/p/24799071>） | ！！！语言不负责内存边界检查。  1、数组下标的取值从0到**元素个数**-1。 |
| 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（最左侧的基本数据类型当作1个\*处理）插入1个\*，占4个字节。<http://blog.csdn.net/daniel_ice/article/details/6857019>   |  | | --- | | #include<stdio.h>  int main(void)  {  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、p2各增加60。 | |

|  |  |
| --- | --- |
|  | 文字 |
| const char[**字符个数**+1] | "(**字符**)\*"  不能在不是\的字符后使用回车。 |
| **某数据类型**的指针 | 0  #include<stdio.h>/#include<stdlib.h>/#include<string.h>/#include<time.h>/#include<wchar.h>   |  |  | | --- | --- | | NULL | 0 | |
| **某数据类型**的数组 | (在**某数据类型**最后1个\*(const )?后（最左侧的基本数据类型当作1个\*处理）插入1个[(**元素个数**)?]){**表达式**(,**表达式**)\*} |

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（其实小括号不需要出现，因为[]优先级更高。）

...\*(const )?**变量名**[**元素个数**]...，数组

若变量是指针，将\*(const )?删去后得到变量指向的数据类型；若变量是数组，将[元素个数]删去后得到每个元素的数据类型。

！！！再强调一遍，所有的\*后都可以带1个const，若是定义声明且最后一个\*（最左侧的基本数据类型当作1个\*处理）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） |

（以**某数据类型**作为返回值数据类型的）函数

与（**某数据类型**的）数组类似：()的优先级比\*高；若变量是函数，将(**形参列表**)删去后得到返回值的数据类型。

（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 | | | 的返回值类型是该类型，形参列表是(void)的函数 | | |
| 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(void);  (int(void))  1个int(void)类型的变量（1个返回值类型是int，形参列表是(void)的变量） | | |
| 的指针 | 的含3个该类型的元素的数组 | 的返回值类型是该类型，形参列表是(void)的函数 | 的指针 | 的含5个该类型的元素的数组 | 的返回值类型是该类型，形参列表是(void)的函数 | 的指针 | 的含3个该类型的元素的数组 | 的返回值类型是该类型，形参列表是(char a1)的函数 |
| 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(void)  (int\*(void))  1个int\*(void)类型的变量（1个返回值类型是int\*，形参列表是(void)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(void)[3];  (int(void)[3])  1个int(void)[3]类型的变量（1个返回值类型是int[3]，形参列表是(void)的变量） | int(\*v)(void);  (int(\*)(void))  1个int(\*)(void)类型的变量（1个储存int(void)类型变量地址的变量） | int v[3](void);  (int[3](void))  1个int[3](void)类型的变量（3个int(void)类型的变量） | int v(char a1)(void);  (int(char)(void))  1个int(char)(void)类型的变量（1个返回值类型是int(void)，形参列表是(char a1)的变量） |
|  |  |  |  |  | 错误，函数返回值类型是数组 | 所谓的函数指针 | 错误，函数的数组 | 错误，函数返回值类型是函数 |

## 数据类型转换

|  |
| --- |
| 数据类型转换  1、提升：char、unsigned char、short自动提升到int；若short和int一样大（unsigned short最大值比int最大值大），则unsigned short自动提升到unsigned int，否则unsigned short自动提升到int。若1个运算符使用了不同级别的数据类型的值，级别低的值自动转换为级别高的值。long double>=double>=float>=unsigned long long>=long long>=unsigned long>=(long,unsigned int)>=int。若int和long一样大（unsigned int最大值比long最大值大），则unsigned int>=long，否则long>=unsigned int。  2、降级：浮点数赋值（包括传参、返回值）给整型左值，截尾取整。 |
| 1、赋值时，非void数据类型的指针可以自动转换为void\*。  2、赋值时，void\*可以自动转换为非void数据类型的指针。 |

## 创建数据类型的别名

|  |  |
| --- | --- |
| 创建别名  typedef **变量引用声明语句**中**变量名**换成**数据类型的别名** | 1、**变量引用声明语句**不初始化。 |
| （使用别名）声明变量  与基本数据类型的[变量的定义声明、引用声明]相同。 | 1、变量的数据类型是在**某数据类型**最后1个\*(const )?后（最左侧的基本数据类型当作1个\*处理）插入1个(**访问限定符** )\*。   |  | | --- | | typedef int (\* t1[1][1])[1][1];  typedef int const(\* t2[1][1])[1][1];  typedef int (\*const t3[1][1])[1][1];  int main(void)  {  t1 a1;  t2 a2;  t3 a3={0};必须初始化  t1 const a4={0};a4的数据类型是int(\*const[1][1])[1][1]，必须初始化  } | |

# 语句

1、编译器支持至少127层嵌套。

|  |  |  |  |
| --- | --- | --- | --- |
| 声明语句 | | | |
| 简单语句 | 空语句 | ; | 1 |
| 表达式语句 | **表达式**; | 1 |
| 结构化语句 | if(**表达式**)  **语句** | 1 |
| if(**表达式**)  **语句**  else  **语句** | 1 |
| switch(**表达式**)  {  ((case **表达式**:|default:)  ( **语句**  )\*)\*} | 1 |
| while(**表达式**)  **语句** | 1 |
| for(**语句**(**表达式1**)?;(**表达式2**)?)  **语句** | 1 |
| do  **语句**  while(**表达式**); | 1 |
| break; | 1 |
| continue; | 1 |
| return( **表达式**)?; | 1 |
| goto **标签名**; | 1 |
| 复合语句（代码块、程序块） | | {  ( **语句**  )\*} | 1 |

控制转移

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 选择结构   |  |  |  | | --- | --- | --- | | if(**表达式**)  **语句** |  |  | | if(**表达式**)  **语句**  else  **语句** | 相当于   |  | | --- | | if(**表达式**)  **语句**  if(!**表达式**)  **语句** |   才怪，因为执行语句可能会影响表达式的值。 |  | | switch(**表达式**)  {  ((case **表达式**:|default:)  ( **语句**  )\*)\*} | 1、switch语句的表达式的数据类型是整型。  2、case的表达式是常量表达式，数据类型是整型。1个switch语句直接包含的每个case标签的表达式的值互不相等。  3、1个switch语句直接包含至多1个default标签。若直接包含了1个default标签，直接包含至少1个case标签。  4、（无论default在什么位置，）若无case标签被匹配，控制跳到default标签。  5、switch语句直接包含的声明语句不能出现在switch语句直接包含的case标签或default标签之前（只能出现在switch语句直接包含的所有case标签和default标签之后）。  6、switch语句直接包含的标签与}之间至少有1条语句。 | 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、表达式可以换成1个变量且初始化的去掉分号的声明语句。 | | for(**语句1**(**表达式1**)?;(**表达式2**)?)  **语句2** | 相当于   |  | | --- | | **语句1**  while(**表达式1**)  {  **语句2**  **表达式2**;  } |   才怪，因为**语句2**可能直接包含continue;。  1、for后的括号内语句是空语句、声明语句或表达式语句。  2、**表达式1**不写相当于写1。  3、**表达式1**可以换成1个变量且初始化的去掉分号的声明语句。 | | do  **语句**  while(**表达式**); |  | |
| 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是void的函数必须包含至少1个return语句。  2、返回值的数据类型是void的函数可以return(void)**表达式**;、return **调用返回值数据类型为void的函数**。 | | goto **标签名**; |  | ！！！default是关键字，不能做标识符，因此不能做标签名。  1、**标签名**:定义1个标签。  2、标签与直接包含它的代码块的}之间至少有1条语句。 | |
| 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。 | |
| 结束程序  #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  #include<stdlib.h>   |  |  | | --- | --- | | EXIT\_SUCCESS或0 | 程序成功结束。 | | EXIT\_FAILURE | 程序未成功结束。 | | 其它值 | 和库的实现方法、系统有关。 | | |

# 函数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 定义声明  将变量名是**函数名**，数据类型是**返回值类型**的（声明1个变量且不初始化的）变量声明语句末尾的;换成：   |  | | --- | | (void)  {  ( **语句**  )\*} | | (**变量声明语句**(,**变量声明语句**)\*(,...)?)  {  ( **语句**  )\*}  每个**变量声明语句**声明1个变量且不初始化。  去掉**变量声明语句**末尾的分号。  **变量声明语句**存储类型可以是auto或register。 | | 1、形参列表是(void)表示不接收实参。  2、不返回值的函数，返回值的数据类型是void。  3、若形参的数据类型是**某数据类型**的数组，表示形参的数据类型是**某数据类型**的指针。因此，数组数据类型的第1维（对应的元素占的字节数最多的1维）的长度可以不写，写的必须是正整数，会被忽略。例：   |  | | --- | | void f(int\*(\*a[3][5])[7]){}  表示（将[3]变成\*，添加括号保证结合顺序不变）  void f(int\*(\*(\*a)[5])[7]){} |  |  | | --- | | 若编译器支持变长数组，可以使用变长数组数据类型，用前面其它形参定义数组每维长度。  若形参数据类型是变长数组，存储类型是auto。 |   4、返回值的数据类型不是数组。 |
| 1、大括号和其中的语句是函数体，函数体前的部分是函数头。函数头;是函数原型。  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); |   ①调用va\_arg会修改va\_list对象，使其指向下一个形参。  ②va\_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。  ③**最后1个普通形参**的存储类型不是register，数据类型不是数组、函数、[与数据类型提升能达到的任何一种数据类型不兼容的数据类型]。  ④由于没有原型声明（(,...)?），主调函数对每个实参进行数据类型提升，见[数据类型提升规则]。  ⑤从基本数据类型到**数据类型**的派生关系不含有[]关系。 |
| 引用声明（函数原型）  在变量名是**函数名**，数据类型是**返回值类型**的变量引用声明语句中的变量名后插入(void)或(**变量声明语句**(,**变量声明语句**)\*(,...)?)。 | 1、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。   |  | | --- | | 若编译器支持变长数组，可以使用变长数组数据类型，用前面其它形参定义数组每维长度。  若形参数据类型是变长数组，存储类型是auto。  若省略前面形参的名称，必须用\*代替省略的维的长度。 | |
| ！！！变量声明语句可以是声明多个变量的。 |
| 调用  **函数名**()  **函数名**(**表达式1**(,**表达式**)\*) | 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：在函数原型不提供形参数据类型的情况下（(,...)?），char、unsigned char、short提升到int；若short和int一样大（unsigned short最大值比int最大值大），则unsigned short提升到unsigned int，否则unsigned short提升到int；float提升到double。  3、指针数据类型传参、返回值的规则都与指针数据类型赋值的规则相同。 |
|  |
| 函数指针  见[（某数据类型的）派生数据类型]。   |  |  |  | | --- | --- | --- | | 变量的定义声明、引用声明  在**函数名**是**变量名**的函数声明中，将**函数名**换成(\***变量名**)，在;前插入(=**表达式**)?。 | | 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、**变量名**的数据类型是**函数头**中**函数名**换成(\_\_cdecl\*)，占4个字节；  &**函数名**的数据类型是**函数头**中**函数名**换成(\_\_cdecl\*)，占4个字节；  **函数名**的数据类型是**函数头**中**函数名**换成(\_\_cdecl\*)，不能使用sizeof；  (\*)+**函数名**的数据类型是**函数头**中去掉**函数名**，不能使用sizeof。  <http://www.cnblogs.com/chinazhangjie/archive/2012/08/18/2645475.html> | | 操作  =赋值  ()函数调用  \*解引用  &取地址  sizeof按字节确定大小  （函数名和指向函数的指针自动互相转换。） | (\***函数指针**)()  (\***函数指针**)(**表达式1**(,**表达式**)\*) | 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**函数头**中**函数名**换成(\_\_cdecl\*)。  2、&**函数指针**的数据类型为**函数头**中**函数名**换成(\_\_cdecl\*\*)。 | | 函数形参的数据类型是函数指针  此时**变量声明语句**是函数指针的声明语句。 | | 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  对[函数指针声明语句]使用[函数定义声明]、[函数引用声明]。 | | 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。（经常在参数等于定义区间的端点时出现。） | 下溢、上溢（整数小于最小值或大于最大值，浮点数量级太小或太大。） | | 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。 |

## 字符转换函数

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

|  |
| --- |
| 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。 |

## 字符映射函数

#include<ctype.h>

|  |
| --- |
| 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。 |

## 字符串转换为数字

#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。 | 每个分隔符是1个字符。  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>

#define EOF (-1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| int scanf(const char\*format,...);  成功，返回被赋值的表达式的个数（可能为0，赋值抑制当然不计入内）；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、转换说明。  文本字符：空格表示忽略至少0个连续出现的空白字符（广义空格）（' '、'\t'、'\n'等）。  转换说明：%(\*)?(域宽)?(长度修饰符)?转换说明符  赋值抑制（滞后赋值）字符：没有，正常；有，从输入流中读取数据并将其丢弃，没有对应的实参。  域宽（最大字段宽度）：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 | | hh (signed)?/unsigned char  h短整型  l长整型  ll (signed)?/unsigned long long | d | 10进制整数 | 指向整型变量的指针 | | i | | 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 |  |   除了%c以外的转换说明符：忽略至少0个连续出现的空白字符->读入至少0个连续出现的非空白字符->读入1个不适合该类型的非空白字符或读入1个空白字符或什么都不读因为达到域宽了->将刚读的1个字符（如果有）放回输入流。若第2步没有适合该类型的非空白字符，则表达式不被赋值，停止读取（若1个表达式被赋值，则它前面的表达式被赋值；若1个表达式不被赋值，则它后面的表达式不被赋值）。  读入下一个非空白字符、连续读入非空白字符且丢弃所有广义空格   |  | | --- | | scanf(" %c",...); |   跳过至少0个连续出现的空白字符与数字字符   |  | | --- | | while(scanf("%d",temp)==1); |   清理输入流   |  |  |  | | --- | --- | --- | | fflush(stdin); | while(getchar()!='\n'); | scanf("%\*s"); | | |
| int printf(const char\*format,...);  成功，返回输出字符个数（结尾的'\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、转换说明。  转换说明：%(标记)?(域宽)?(.精度)?(长度修饰符)?转换说明符  标记（标志）   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | | | 有 | | 没有 | | | - | | | 在域宽内左对齐打印  覆盖0 | | 在域宽内右对齐打印 | | | + | | | 在正数前面显示1个正号  覆盖1个空格 | | 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（字段宽度）：总字符数  1、负号占用1位域宽。  2、域宽小于实际字符数，自动扩展。  精度   |  |  |  |  | | --- | --- | --- | --- | | 转  换  说  明  符 | 整型转换说明符 | 数字位数小于精度，数值前加前缀0。  覆盖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，当然是从左侧开始截取。 | |   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 | | hh (signed)?/unsigned char  h短整数  l长整数  ll (signed)?/unsigned long long | d | 10进制整数 | | 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 | | a | 以十六进制形式显示浮点数，p | | E | 以指数形式显示浮点数，E | | A | 以十六进制形式显示浮点数，P | | 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 | 打印字符 | | 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 |   整数流程：整数->根据精度补0->负数加负号，正数根据标志补正号或空格->放置在域宽中->（没有左对齐且填充0且没使用精度（且域宽有剩余））在数字前面插入0补域宽   |  | | --- | | 域宽补0和精度补0  #include<stdio.h>  int main(void)  {  printf("[%d]\t[%8d]\t[%.8d]\n",1234,1234,1234);  printf("[%d]\t[%8d]\t[%.8d]\n",-1234,-1234,-1234);  printf("[%-d]\t[%-8d]\t[%-.8d]\n",1234,1234,1234);  printf("[%-d]\t[%-8d]\t[%-.8d]\n",-1234,-1234,-1234);printf("\n");  printf("[%+d]\t[%+8d]\t[%+.8d]\n",1234,1234,1234);  printf("[%+d]\t[%+8d]\t[%+.8d]\n",-1234,-1234,-1234);  printf("[%-+d]\t[%-+8d]\t[%-+.8d]\n",1234,1234,1234);  printf("[%-+d]\t[%-+8d]\t[%-+.8d]\n",-1234,-1234,-1234);printf("\n");  printf("[% d]\t[% 8d]\t[% .8d]\n",1234,1234,1234);  printf("[% d]\t[% 8d]\t[% .8d]\n",-1234,-1234,-1234);  printf("[%- d]\t[%- 8d]\t[%- .8d]\n",1234,1234,1234);  printf("[%- d]\t[%- 8d]\t[%- .8d]\n",-1234,-1234,-1234);printf("\n");  printf("[%0d]\t[%08d]\t[%0.8d]\n",1234,1234,1234);  printf("[%0d]\t[%08d]\t[%0.8d]\n",-1234,-1234,-1234);  printf("[%-0d]\t[%-08d]\t[%-0.8d]\n",1234,1234,1234);  printf("[%-0d]\t[%-08d]\t[%-0.8d]\n",-1234,-1234,-1234);printf("\n");  printf(" +0\n");  printf(" [%+0d]\t[%+08d]\t[%+0.8d]\n",1234,1234,1234);  printf(" [%+0d]\t[%+08d]\t[%+0.8d]\n",-1234,-1234,-1234);  printf(" [%-+0d]\t[%-+08d]\t[%-+0.8d]\n",1234,1234,1234);  printf(" [%-+0d]\t[%-+08d]\t[%-+0.8d]\n",-1234,-1234,-1234);  printf(" \_0\n");  printf(" [% 0d]\t[% 08d]\t[% 0.8d]\n",1234,1234,1234);  printf(" [% 0d]\t[% 08d]\t[% 0.8d]\n",-1234,-1234,-1234);  printf(" [%- 0d]\t[%- 08d]\t[%- 0.8d]\n",1234,1234,1234);  printf(" [%- 0d]\t[%- 08d]\t[%- 0.8d]\n",-1234,-1234,-1234);printf("\n");  } | | 域宽、精度不足  #include<stdio.h>  int main(void)  {  printf("[%d]\t[%2d]\t[%.2d]\n",1234,1234,1234);  printf("[%d]\t[%2d]\t[%.2d]\n",-1234,-1234,-1234);  printf("[%-d]\t[%-2d]\t[%-.2d]\n",1234,1234,1234);  printf("[%-d]\t[%-2d]\t[%-.2d]\n",-1234,-1234,-1234);printf("\n");  printf("[%+d]\t[%+2d]\t[%+.2d]\n",1234,1234,1234);  printf("[%+d]\t[%+2d]\t[%+.2d]\n",-1234,-1234,-1234);  printf("[%-+d]\t[%-+2d]\t[%-+.2d]\n",1234,1234,1234);  printf("[%-+d]\t[%-+2d]\t[%-+.2d]\n",-1234,-1234,-1234);printf("\n");  printf("[% d]\t[% 2d]\t[% .2d]\n",1234,1234,1234);  printf("[% d]\t[% 2d]\t[% .2d]\n",-1234,-1234,-1234);  printf("[%- d]\t[%- 2d]\t[%- .2d]\n",1234,1234,1234);  printf("[%- d]\t[%- 2d]\t[%- .2d]\n",-1234,-1234,-1234);printf("\n");  printf("[%0d]\t[%02d]\t[%0.2d]\n",1234,1234,1234);  printf("[%0d]\t[%02d]\t[%0.2d]\n",-1234,-1234,-1234);  printf("[%-0d]\t[%-02d]\t[%-0.2d]\n",1234,1234,1234);  printf("[%-0d]\t[%-02d]\t[%-0.2d]\n",-1234,-1234,-1234);printf("\n");  printf(" +0\n");  printf(" [%+0d]\t[%+02d]\t[%+0.2d]\n",1234,1234,1234);  printf(" [%+0d]\t[%+02d]\t[%+0.2d]\n",-1234,-1234,-1234);  printf(" [%-+0d]\t[%-+02d]\t[%-+0.2d]\n",1234,1234,1234);  printf(" [%-+0d]\t[%-+02d]\t[%-+0.2d]\n",-1234,-1234,-1234);  printf(" \_0\n");  printf(" [% 0d]\t[% 02d]\t[% 0.2d]\n",1234,1234,1234);  printf(" [% 0d]\t[% 02d]\t[% 0.2d]\n",-1234,-1234,-1234);  printf(" [%- 0d]\t[%- 02d]\t[%- 0.2d]\n",1234,1234,1234);  printf(" [%- 0d]\t[%- 02d]\t[%- 0.2d]\n",-1234,-1234,-1234);printf("\n");  } | | [精度]覆盖[填充0]  #include<stdio.h>  int main(void)  {  printf("[%0d]\t[%016.8d]\t[%08.16d]\n",1234,1234,1234);  printf("[%0d]\t[%016.8d]\t[%08.16d]\n",-1234,-1234,-1234);  } | | |
| ！！！转换说明符决定数据的解释方式，可以和数据本来的类型不匹配   |  |  |  | | --- | --- | --- | | 数据类型 | scanf | printf | | long double长双精度浮点数 | %Lf、%Le、%LE、%Lg、%LG | | | double双精度浮点数 | %lf、%le、%lE、%lg、%lG | %f、%e、%E、%g、%G | | float浮点数 | %f、%e、%E、%g、%G | | | unsigned long long int | %llu | | | long long int | %lld | | | 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 | |   printf打印long double还可以用%La、%LA，打印double、float还可以用%a、%A。 | |
| 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个非负数；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'。 |

#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'。 | |
| int kbhit(void);  若当前有键盘输入，返回非0值；否则返回0。  是非阻塞函数。 | |

# 施工区域

|  |
| --- |
|  |

将缓冲区内容传送到屏幕：

|  |
| --- |
| 缓冲区满 |
| 遇到换行符 |
| 需要输入 |

#include<wchar.h>

WCHAR\_MIN

WCHAR\_MAX

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| #include<limits.h>   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | char | CHAR\_BIT  bit数 | SCHAR\_MIN | SCHAR\_MAX | UCHAR\_MAX | CHAR\_MIN  SCHAR\_MIN或0 | CHAR\_MAX  SCHAR\_MAX或UCHAR\_MAX | | wchar\_t | MB\_LEN\_MAX  bit数 |  |  |  |  |  | | short |  | SHRT\_MIN | SHRT\_MAX | USHRT\_MAX |  |  | | int |  | INT\_MIN | INT\_MAX | UINT\_MAX |  |  | | long |  | LONG\_MIN | LONG\_MAX | ULONG\_MAX |  |  | | long long |  | LLONG\_MIN | LLONG\_MAX | ULLONG\_MAX |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | #include<stdint.h> | | | | #include<inttypes.h>  **x**：(d|i|o|u|x|X)  **N**：(8|16|32|64) | | | 长度最长 | intmax\_t |  |  |  | PRI**x**MAX | SCN**x**MAX | | uintmax\_t |  |  |  | | 确切长度类型 | int8\_t | int16\_t | int32\_t | int64\_t | PRI**xN** | SCN**xN** | | uint8\_t | uint16\_t | uint32\_t | uint64\_t | | 最小长度类型 | int\_least8\_t | int\_least16\_t | int\_least32\_t | int\_least64\_t | PRI**x**LEAST**N** | SCN**x**LEAST**N** | | uint\_least8\_t | uint\_least16\_t | uint\_least32\_t | uint\_least64\_t | | 最快最小长度类型 | int\_fast8\_t | int\_fast16\_t | int\_fast32\_t | int\_fast64\_t | PRI**x**FAST**N** | SCN**x**FAST**N** | | uint\_fast8\_t | uint\_fast16\_t | uint\_fast32\_t | uint\_fast64\_t | |  | intptr\_t |  |  |  | PRI**x**PTR | SCN**x**PTR | |  | uintptr\_t |  |  |  | |
| #include<float.h>   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | 尾数位数 |  |  |  | 最小正数 |  | 大于1的最小数与1的差 | | float | FLT\_MANT\_DIG | FLT\_DIG | FLT\_MIN\_EXP  FLT\_MIN\_10\_EXP | FLT\_MAX\_EXP  FLT\_MAX\_10\_EXP | FLT\_MIN | FLT\_MAX | FLT\_EPSILON | | double | DBL\_MANT\_DIG | DBL\_DIG | DBL\_MIN\_EXP  DBL\_MIN\_10\_EXP | DBL\_MAX\_EXP  DBL\_MAX\_10\_EXP | DBL\_MIN | DBL\_MAX | DBL\_EPSILON | | long double | LDBL\_MANT\_DIG | LDBL\_DIG | LDBL\_MIN\_EXP  LDBL\_MIN\_10\_EXP | LDBL\_MAX\_EXP  LDBL\_MAX\_10\_EXP | LDBL\_MIN | LDBL\_MAX | LDBL\_EPSILON |   FLT\_RADIX（基数）、FLT\_ROUNDS（如何舍入，enum）、FLT\_EVAL\_METHOD（如何求表达式的值，enum）、DECIMAL\_DIG  <https://stackoverflow.com/questions/39703648/what-is-the-difference-between-decimal-dig-and-ldbl-dig-in-float-h> |

操纵流的函数

<http://www.cnblogs.com/QJohnson/archive/2011/06/24/2089414.html>

<http://linux.chinaunix.net/techdoc/system/2008/08/19/1026178.shtml>

<http://unix.stackexchange.com/questions/28384/how-can-same-fd-in-different-processes-point-to-the-same-file>

<http://unix.stackexchange.com/questions/195057/what-is-an-open-file-description/195164#195164>

1、只有系统内核有1个打开文件列表。1个FILE结构体对应1个文件，FILE结构体的1个成员储存文件描述符。当1个文件的所有实例都关闭之后，内核释放这个数据结构。

2、1个进程对应1个文件描述符表，不同进程的指向同一个文件的文件描述符可能不相等，不同进程的相等的文件描述符可能指向不同的文件。

3、程序使用指向FILE结构体的指针操纵文件。1个FILE结构体对应1个流。

4、程序开始执行时，自动打开标准输入流、标准输出流、标准错误流，自动声明指向它们的指针stdin、stdout、stderr，对应的文件描述符分别是0、1、2。

杂项

#include<stdio.h>

|  |
| --- |
| int fileno(FILE\*stream);  成功，返回stream的文件描述符。 |

将流与文件关联（**流的读写权**、**流的打开模式**）

#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是文件描述符。  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）

|  |  |
| --- | --- |
| 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'或读入n-1个字符，停止。  '\n'被丢弃。  末尾添加'\0'。 | int fputs(const char\*s,FILE\*stream);  成功，返回1个非负数；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。 |

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。 |

**流的指示器**

文件读写位置（位置指示器）

#include<stdio.h>

|  |
| --- |
| 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（错误指示器）（可以调用clearerr,rewind或freopen清除）

文件结束标志（文件结束指示器）（可以调用clearerr,freopen,rewind,fseek或fsetpos清除）

#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）。 |

**流的缓冲区**

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

#include<stdio.h>

|  |
| --- |
| 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）。 |

**流的编码定位**

1、对流进行的第1次io操作决定流的编码定位。

#include<wchar.h>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| int fwide(FILE\*stream,int mode);  调用后流没有编码定位，返回0；调用后流是面向字节的，返回1个负数；调用后流是面向宽字符的，返回1个正数。  不能改变1个已经有编码定位的流的编码定位。  调用freopen后可以改变流的编码定位。  mode   |  |  | | --- | --- | | 0 | 不改变流的编码定位 | | 负数 | 面向字节 | | 正数 | 面向宽字符 | |

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。 | |

错误用词

What is a multibyte character set?<http://stackoverflow.com/questions/748965/what-is-a-multibyte-character-set>

What is ANSI format?<http://stackoverflow.com/questions/701882/what-is-ansi-format>

<http://www.fmddlmyy.cn/text6.html>

UTF-8 Everywhere<http://utf8everywhere.org/>

<https://en.wikipedia.org/wiki/Variable-width_encoding>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字符集 | UCS | | | | |
| 编码方案 | UTF-8、 | UTF-16 | UTF-16 | UTF-32 |  |
|  | char  '**字符**'  "(**字符**)\*" | wchar\_t（是某整型的别名）  L'**字符**'  L"(**字符**)\*" | char16\_t（是某无符号整型的别名） | char32\_t（是某无符号整型的别名） | TCHAR（是char或wchar\_t的别名，取决于工程设置）  \_T("(**字符**)\*")、\_TEXT("(**字符**)\*") |
|  | int | wint\_t（是某整型的别名） |  |  | TINT（是int或wint\_t） |
|  |  |  | \_\_STD\_UTF\_16\_\_ | \_\_STD\_UTF\_32\_\_ |  |
|  | EOF | WEOF |  |  |  |
| 字符占的字节数 |  | mblen  mbrlen |  |  |  |
| 字符类型转换 |  | btowc |  |  |  |
|  | wctob |  |  |  |
|  | mbtowc  mbrtowc | mbrtoc16 | mbrtoc32 |  |
|  | wctomb  wcrtomb | c16rtomb | c32rtomb |  |
| 字符串类型转换 |  | mbstowcs  mbsrtowcs |  |  |  |
|  | wcstombs  wcsrtombs |  |  |  |
| 转换状态 |  | mbsinit |  |  |  |
|  | memset | wmemset |  |  |  |
| memcpy | wmemcpy |  |  |  |
| memmove | wmemmove |  |  |  |
| memcmp | wmemcmp |  |  |  |
| memchr | wmemchr |  |  |  |
|  | isxdigit | iswxdigit |  |  |  |
| isalnum | iswalnum |  |  |  |
| isdigit | iswdigit |  |  |  |
| isalpha | iswalpha |  |  |  |
| islower | iswlower |  |  |  |
| isupper | iswupper |  |  |  |
| iscntrl | iswcntrl |  |  |  |
| isprint | iswprint |  |  |  |
| isgraph | iswgraph |  |  |  |
| ispunct | iswpunct |  |  |  |
| isspace | iswspace |  |  |  |
| isblank | iswblank |  |  |  |
| tolower | towlower |  |  |  |
| toupper | towupper |  |  |  |
|  | atoi | / |  |  |  |
| atol | / |  |  |  |
| atof | / |  |  |  |
| strtol | wcstol |  |  |  |
| strtoul | wcstoul |  |  |  |
| strtod | wcstod |  |  |  |
|  | strcpy | wcscpy |  |  |  |
| strncpy | wcsncmp |  |  |  |
| strcat | wcscat |  |  |  |
| strncat | wcslen |  |  |  |
| strxfrm | wcsxfrm |  |  |  |
| strcmp  strcoll | wcscmp  wcscoll |  |  |  |
|  |  |  |
| strncmp | wcsncmp |  |  |  |
| strchr | wcschr |  |  |  |
| strrchr | wcsrchr |  |  |  |
| strpbrk | wcsncmp |  |  |  |
| strstr | wcsstr |  |  |  |
| strspn | wcsspn |  |  |  |
| strcspn | wcscspn |  |  |  |
| strtok | wcstok |  |  |  |
| strerror | / |  |  |  |
| strlen | wcslen |  |  |  |
|  | sscanf | swscanf |  |  |  |
| sprintf | / |  |  |  |
| snprintf | swprintf |  |  |  |
|  | getc | getwc |  |  |  |
| ungetc | ungetwc |  |  |  |
| putc | putwc |  |  |  |
| fgetc | fgetwc |  |  |  |
| fputc | fputwc |  |  |  |
| getchar | getwchar |  |  |  |
| putchar | putwchar |  |  |  |
| gets |  |  |  |  |
| puts |  |  |  |  |
| fgets | fgetws |  |  |  |
| fputs | fputws |  |  |  |
| scanf | wscanf |  |  |  |
| printf | wprintf |  |  |  |
| fscanf | fwscanf |  |  |  |
| fprintf | fwprintf |  |  |  |

C programming: How to program for Unicode?<http://stackoverflow.com/questions/526430/c-programming-how-to-program-for-unicode>

Is TCHAR still relevant?<http://stackoverflow.com/questions/234365/is-tchar-still-relevant/3002494#3002494>

UTF-8 or UTF-16 or UTF-32 or UCS-2<http://stackoverflow.com/questions/3473295/utf-8-or-utf-16-or-utf-32-or-ucs-2/>

【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>