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>

更新地址：

若发现文档内容与实际不符，请向GitHub上的jhcarl0814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>

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、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，因此是否去求表达式的值也需要被关注，例如：调用函数进而变量的值被修改。）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ()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。 | 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、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）。  2、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)，表达式2的类型为整型。 |
| .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、表达式的类型为整型。 |
| (**类型名**)C风格的一元强制类型转换 | (**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类型的值不能转换为其它类型的值。 |
| \*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 |
| &取地址 | &**表达式** | 得到操作数的地址。 | 1、表达式的值是左值 |
| sizeof按字节确定大小 | sizeof(**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、表达式的类型都为整型。 | 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、表达式的类型都为整型。 |
| |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、表达式的类型都为整型。 |
| &&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** |  |  |
| %=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）/右（左等于0） |
| 其它运算符 | undefined |

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 | 在所有函数外定义 | 从声明到程序结束被访问 |  |
| 程序块作用域 | 局部变量、函数的形参、函数内的函数原型 | 在程序块内定义 | 从声明到程序块结束被访问 | 程序块嵌套且内外标识符重名时，在且仅在内标识符能被访问的区域（←从声明到程序块结束被访问）外标识符不能被访问。 |
| 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）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| auto | **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、赋值之前变量的值为undefined。 |
| 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、赋值之前变量的值为undefined。  2、变量不能取地址  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。 |

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 |  | | |
| 定义变量  (**访问限定符** )\***类型名** **变量名**(=**表达式**)?(,**变量名**(=**表达式**)?)\*; | 1、若访问限定符列表中含有const则必须初始化。  2、const **类型名**和**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 | undefined behavior | | | 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

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| （**某类型**的）指针   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 定义变量  在**某类型**的变量定义语句中将**变量名**(=**表达式**)?换为\*(const )?**变量名**(=**表达式**)?，如果变量名后紧接着[则需要加1层括号（因为[]优先级更高）。  例外：可以定义void\*(const)?类型的指针变量。 | | ！！！新类型还可以继续用作**某类型**。  1、新类型名是在**某类型**最后1个\*后（如果没有\*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（因为[]优先级更高）。新类型占4个字节。  2、若最后一个\*后有const，则必须初始化。 | 1、\*后有const表示该变量解引用得到的值是不能修改的。   |  | | --- | | 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** | 1、void\*类型的值不能进行和加减有关的操作。  2、非void\*类型的值赋给void\*类型的变量不需要强制类型转换运算符。  3、若最后一个\*后有const则必须初始化。  4、能不通过强制类型转换运算符就赋给指针的常量只有0和NULL。  5、表达式的类型为整型。  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(**某类型**)。  2、指针-指针得到的值是(指针的值-指针的值)/sizeof(**某类型**)。两个指针指向同一数组的元素时才有意义。  3、**某类型**\*类型的变量=&**某类型**的变量;。  4、**某类型**\*类型的变量->成员名等价于  \*(**某类型**\*类型的变量).成员名。  5、\*p+=1等价于(\*p)++。 | |
| （**某类型**的）数组   |  |  |  | | --- | --- | --- | | 定义变量  在**某类型**的变量定义语句中将**变量名**(=**表达式**)?换为**变量名**[**元素个数**](={(**表达式**(,**表达式**)\*)?})?或  **变量名**[]={**表达式**(,**表达式**)\*}。 | ！！！新类型还可以继续用作**某类型**。  1、新类型名是在**某类型**最后1个\*后（如果没有\*则是第1个[前）插入1个[**元素个数**]。新类型占sizeof(**某类型**)\***元素个数**个字节。  2、若初始化且初始值个数小于成员个数，剩余元素：非指针初始化为0，指针初始化为NULL。  3、如果变量名左端是\*（即**某类型**名左端是\*），应继续将**变量名**换为(**变量名**)；新类型名是**某类型的基础类型** (**某类型中间的剩余部分**)[**元素个数**]。 | 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、数组名取地址<http://blog.csdn.net/daniel_ice/article/details/6857019>  int c[3][4],(\*p1)[4],(\*p2)[3][4];  p1=c;p2=&c;  printf("%p %p\n%p %p\n%p %p\n%p %p\n"  ,c,c+1,&c,&c+1,p1,p1+1,p2,p2+1);  结果：c、p1各增加16，&c、p1各增加48。 | |
| 判断变量是指针还是数组：看最靠近**变量名**的符号  (\***变量名**)，指针  (**变量名**[元素个数])，数组  **变量名**左边连着\*，右边连着[，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。   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | int v;  (int)  1个int类型的变量 | | | | | | | | | 的指针 | | | | 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 | | 含3个该类型的元素的数组 | | 的指针 | | 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 | 含3个该类型的元素的数组 | 的指针 | 含5个该类型的元素的数组 | 的指针 | 含5个该类型的元素的数组 | 的指针 | 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   |  |  |  | | --- | --- | --- | | 声明类型  struct( **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 **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，变量的类型为整型。  2、成员的类型不是当前正在声明的类型。（但可以是当前正在声明的类型的派生类型，派生关系含有至少1个\*关系（即不能光有[]关系）。因为\*关系会把成员类型变成**某类型**的指针，大小由未知转为已知或保持已知。例如：  struct **类型名** **成员名**[1][1][1][1];错误。  struct **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  struct **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  sizeof按字节确定大小  .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。 | | 在声明类型时定义变量  struct( **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 **类型名**。 | | 在声明类型时使用typedef  typedef struct( **类型名**)?  {  (**变量定义语句**  )\*  }**类型名**(,**类型名**)\*; | |  | |
| 共用体   |  |  |  | | --- | --- | --- | | 声明类型  union( **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  union **类型名 变量名**(={(**表达式**)?})?; | | 1、若表达式的值的类型与第1个成员的类型不同，则表达式的值被转换为第1个成员的类型。浮点数转整数，舍去小数部分。 | | 操作  =赋值  &取地址  sizeof按字节确定大小  .访问成员  sizeof按字节确定大小 | | 1、不能用==、!=比较两个共用体。 | | 在声明类型时定义变量  与结构体类似。 | |  | | 在声明类型时使用typedef  与结构体类似。 | |  | |
| 枚举类型   |  |  |  | | --- | --- | --- | | 声明  enum **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、表达式类型为整型，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  enum **类型名 变量名**(=**标识符**)?;  enum **类型名** **变量名**=(enum **类型名**)**表达式**; | | 1、标识符不能使用其它enum类型的标识符。  2、表达式的值可以超出enum类型含有的标识符的值的范围。  3、表达式的值的类型为该类型时，可以省略强制类型转换运算符。 | | 在声明类型时定义变量  与结构体类似。 | |  | | 在声明类型时使用typedef  与结构体类似。 | |  | |

创建类型的别名

|  |
| --- |
| 创建别名  typedef **变量定义语句**中**变量名**换成**新的类型名** |
| 定义变量  (**访问限定符** )\***新的类型名** **变量名**(=**表达式**)?(,**变量名**(=**表达式**)?)\*; |

控制转移

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 选择结构专用   |  |  |  | | --- | --- | --- | | 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。  2、语句数等于0时可以将{}换为;。 |  | | 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。  2、语句数等于0时可以将{}换为;。 |  | | switch(**表达式**)  {  (case **表达式**:  **语句**  )+} | 1、switch的表达式类型为整型。  2、case的表达式类型为整型，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。 | |
| 循环结构专用   |  |  | | --- | --- | | 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。  2、语句数等于0时可以将{}换为;。 | | 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、大括号内的语句数等于0时可以将{}换为;。  3、for后的括号内不出现其它的分号。  4、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。  2、语句数等于0时可以将{}换为;。 | |
| 跳转专用   |  |  |  | | --- | --- | --- | | 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。  在返回值类型不为void的函数中，返回表达式的值。 | 1、返回值类型不为void的函数必须包含至少1个return语句。  2、返回值类型为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。 | |

函数

|  |  |
| --- | --- |
| 定义  (**访问限定符** )\***返回值类型** **函数名**(void)  {  **语句**  }  (**访问限定符** )\***返回值类型** **函数名**(**形参类型** **形参名**(,**形参类型** **形参名**)\*(,...)?)  {  **语句**  } | 1、返回值类型不为数组。  2、不返回值的函数，返回值类型为void。  3、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>   |  | | --- | | va\_list ap;  va\_start(ap,**最后1个普通形参名**);  (va\_arg(ap,**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类型提升：float类型的实际参数将提升到double；char、short和相应的signed、unsigned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、返回值类型不为void的函数返回1个值。  2、类型提升规则：在没有函数原型的情况下，char与short类型都将被转换为int类型，float类型将被转换为double类型。 |
|  |
| 函数指针   |  |  |  | | --- | --- | --- | | 定义  **返回值类型** (\***变量名**)(void)(=**表达式**)?;  **返回值类型** (\***变量名**)(**形参类型** **形参名**(,**形参类型** **形参名**)\*(,...)?)(=**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、**返回值类型** **变量名**(**形参列表**)=**函数名**;和**返回值类型** **变量名**(**形参列表**)=\***函数名**;是错误的，不存在将1个值转换为函数类型的值的转换过程。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)类型转换为int(int)类型  int b(int)=\*f;错误，不能把int(int)类型转换为int(int)类型  } |   2、**变量名**的类型是**返回值类型** (\_\_cdecl \*)(**形参类型列表**)，占4个字节；  &**函数名**的类型是**返回值类型** (\_\_cdecl \*)(**形参类型列表**)，占4个字节；  **函数名**的类型是**返回值类型** (\_\_cdecl \*)(**形参类型列表**)，不能使用sizeof；  (\*)+**函数名**的类型是**返回值类型** (**形参类型列表**) ，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**的类型为**返回值类型** (\_\_cdecl \*)(**形参类型列表**)。  2、&**函数指针**的类型为**返回值类型** (\_\_cdecl \*\*)(**形参类型列表**)。  3、**返回值类型** (\_\_cdecl \*)(**形参类型列表**)、**返回值类型** (\_\_cdecl \*\*)(**形参类型列表**)、**返回值类型** (**形参类型列表**)类型的值都可以转换为void\*类型的值，但没有任何通过void\*类型的值调用函数的方法（因为不存在将1个值转换为函数类型的值的转换过程）。 | | 函数形参类型是函数指针  **形参类型** **形参名**换成**指向的函数的返回值类型** (\***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**。 | |  | | 函数返回值类型是函数指针  **返回值类型** **函数名**(**形参列表**)换成  **指向的函数的返回值类型**(\***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。 | |  | | |

字符处理函数#include<ctype.h>

|  |
| --- |
| 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 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\*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的指针或NULL。 |  |
| char\*strrchr(const char\*s,int c);  返回s中最后1个c的指针或NULL。 |  |
| char\*strpbrk(const char\*s1,const char\*s2);  返回s1中第1个[s2中任意字符]的指针或NULL。 |  |
| char\*strstr(const char\*s1,const char\*s2);  返回s1中第1个s2的指针或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，返回指向当前标号指针；  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>

|  |  |
| --- | --- |
| sscanf | sprintf |

标准输入输出库函数#include<stdio.h>

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| int scanf(const char\*format,...);  成功，返回被赋值的参数的个数；发生错误，设置错误标识（通过ferror()检测），返回 EOF；遇到结束符，设置文件结束标识（通过feof()检测），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 | 指向无符号整形变量的指针 | | 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 | 指向浮点数类型变量的指针 | |  | c | 字符 | 指向字符类型变量的指针 | | s | 字符串，自动加上'\0'。 | 指向字符类型变量的指针 | | [(**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；发生错误，设置文件错误标志（通过ferror()检测），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 | 对应实参类型为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，对应实参类型为int\*。 | | % | 显示1个百分号 | | |
| int getchar(void);  成功，返回字符的ASCII码；出错或遇到文件结束，返回EOF。 | int putchar(int c);  成功，返回c；出错，返回EOF。 |
| gets | puts |
|  | |
|  | |
|  | |
|  | |
|  | |

|  |  |
| --- | --- |
| int fscanf(FILE\*stream,const char\*format,...);  成功，返回被赋值的参数的个数（可能是0）；读取发生错误，设置错误标识（通过ferror()检测），返回 EOF；遇到结束符，设置文件结束标识（通过feof()检测），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（通过ferror()检测），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。 |
| int getc(FILE\*stream);  成功，返回字符的ASCII码；出错或遇到文件结束，返回EOF。 | int putc(int c,FILE\*stream);  成功，返回c；出错，返回EOF。 |
| fgetc | fputc |
| fgets | fputs |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |