

第 10 章 附录

10.1 关于程序设计的学习方法

掌握程序设计的学习方法，对初学者来说尤其重要。

首先，我们了解学习程序设计的 6 个层次，如下表所示。

层次	比喻	特点	心手状态	说明
0	纸上谈兵	能看懂书，能听懂课，但没有动手编程。	心到手未到	学编程不看书肯定不行，但只看不练是空把式。
1	鹦鹉学舌	思路是别人的，代码也是别人的。脱离样板代码，无法写出程序。	心未到手才来	刚接触的新的程序，编程基本等于“打字”。
2	依葫芦画葫芦	思路是别人的，代码也是别人的。脱离样板代码，可以重现样板程序。	心生手生	能理解别人思路和代码，能重现别人的代码。
3	依葫芦画瓢	思路是别人的，但是代码是自己的，自己的实现代码和样板代码不一样。	心半生手半熟	能理解别人思路和代码，能以自己的方式写代码。
4	蹒跚学步	有思路但是用程序表达出来有困难。	心到手不顺	初步具备独立编程解决问题的能力。
		对于简单问题，思路是自己的，代码也是自己的。	心到手到	
5	自主行走想到哪就能走到哪	建立了计算思维，能独立分析问题，解决问题。算法思路想得到，程序代码写得出。	心熟手熟	能解决大部分实际问题。只求能解决问题，不考虑程序的开发和运行效率。
6	健步如飞运用自如	应用开发方向：能设计和开发应用软件系统，满足实际生产生活的需求。	心灵手巧	以满足实际需求为目标，重视软件工程的方法。追求目标是：开发快速、运行可靠、用户满意。
		算法研究方向：能对复杂问题进行算法设计和分析，能对大数据量的问题能提高程序的效率。	心手合一	重视程序中算法和数据结构，追求目标是程序运行快，省内存。

对于将程序设计作为专业基础的学生来说，应该达到的层次是介于层次 5 和层次 6 之间，也就是说能编程解决实际问题，掌握简单的算法(交换数据、累加、计数、设标记、查表、排序)，初步掌握提高程序的运行效率的方法(减少重复计算，折半查找)，初步接触软件工程的思想(函数的设计、结构体)。

学习程序设计的三种境界：一、照抄现成（教材上或者网上的）的程序；二、理解别人的程序但是自己不能自主修改程序；三、理解并能自主修改程序；五、完全自主地设计程序。

也就是说，直接利用教材或书中的实例代码，这是编程的最低境界。

虽然本书所有编程实例都提供了源代码，但是，鼓励读者用多多尝试用不同的代码、不同的算法、自主设计、亲自动手动脑写程序。每个编程任务对需要解决的问题进行了尽可能详细、准确的描述。因此，对于每个编程任务，编程的目标十分明确----满足编程任务提出的需求。

解决同一个问题（或编程任务）的程序代码远远不止一种写法，首先是算法可以不同，即使算法相同，实现细节也可以不同。大力提倡读者通过对本书的学习后，对每个编程任务用属于自己的思路、算法或程序代码。而不仅仅是将本书中的代码照搬，运行得到正确结果后，就认为自己掌握了，浅尝辄止。应该进一步思考，还有哪些写法可让程序实现同样的功能，这些写法之间有何区别，各有何优缺点等等.....。

对于 C 语言的语法，鼓励大家“知法犯‘法’，以身试‘法’”，也就是编写小程序去尝试和验证 C 语言语法，以加深对 C 语言的理解和体会。只有通过“多试”即“多试验，多尝试，多实践”才能真正掌握 C 语言，做到知其然也知其所以然。“所写即所想，所欲即所得”

（What you write is what you think, and what you want is what you get!）是程序设计需要达到的学习目的。

关于如何学习程序设计，有如下建议：

（1）以解决实际问题为目标。实际的应用问题当然不仅仅是本书作为例题的编程任务，终极目标是解决您自己在学习、工作和生活中问题。

（2）培养对于实际问题的独立分析能力，培养计算思维，以便将自己的解决问题的思路编程能让计算机执行的程序。

（3）对于初学者来说，用程序表达自己的思想需要过“语言关”----设计“程序代码”以实现自己想要的功能。

（4）学习过程必须“动手”（上机编程）和“动脑”（积极地思考）相结合，程序设计是理论与实践的结合。成为编程高手没有捷径。只看书不动手编程，只听讲不动手编程，只看视频不动手编程的人是难以学好程序设计的。只敲代码而不能理解代码背后的原理和算法，这种只重实践而没有理论指导和提高的做法，也很难成为编程高手。

（5）对于已解决的问题，可利用发散思维，试着从以下几个方面进一步考虑：

（a）还有其它方法求解此问题吗？

（b）程序代码还可以写的更加精简吗？

（c）如果改变了输入数据或者问题的条件，应该如何解决呢？例如数据范围变大，数据量变大，要求在更小的内存空间或更短的时间内得到运行结果。

（d）我能将程序的思路用文字、图表等一切可以利用的方式清晰地表达吗？

（e）能否尽量减少重复计算，提高程序的运行效率。

（f）对于这些不同的程序如何衡量它们的运行性能（从运行时间和所需内存空间）进行评价。

10.2 关于在线练习

如何利用程序设计在线裁判系统（Online Judge，简称 OJ）进行练习？

对于学习 C 程序设计的读者来说，是在 OJ 上练习编程是非常有效地提高实际编程能力的手段。虽然国内外各大 OJ 系统（例如北大 OJ、浙大 OJ、杭电 OJ 等）的本来是为程序设计竞赛和训练用，但是，因为该系统有全天候 24 小时在线、全自动裁判、结果客观、准确、快速等特点，已经被越来越多的程序设计学习者和教师用作 C 语言的编程练习平台。

使用的 OJ 的大致步骤如下：

- （1）登：登录 OJ 网站。
- （2）阅：从 OJ 的 Problem Set 中查看需要完成的编程任务。
- （3）想：理解题意。建议用草稿纸和笔各种写写画画，使思路和算法明确。
- （4）写：打开 C 语言编程的 IDE（例如 Code::Blocks），写代码。
- （5）测：调试和测试程序。一定要全面理解任务要求，充分测试。
- （6）交：将程序代码复制后，点击编程任务网页中的 submit 按钮提交到 OJ。
- （7）判：查看 OJ 返回的裁判结果。只有结果为 Accepted 才表示提交程序正确。

本书中的“编程任务”可在湖南农业大学 OJ（<http://210.43.224.19/oj>）上提交。

常见 OJ 系统常见裁判结果

判题结果	简称缩写	判题结果 字面意思	提交到判题服务器的程序 可能存在的问题
Accepted	AC	程序正确	提交的程序正确。
Wrong Answer	WA	答案错误	提交的程序对于某些测试用例输出了错误结果。
Presentation Error	PE	表示错误	提交的程序输出格式（例如空格、回车、跳格）不符合编程任务的输出格式要求。
Compile Error	CE	编译错误	提交的程序存在编译错误。
Runtime Error	RE	运行时错	提交的程序在裁判服务上的运行过程中遇到了异常，导致程序终止。有可能是程序中的数组越界、指针地址错误导致按该地址间接访问时异常。
Time Limit Exceed	TLE	超过时限	提交的程序在裁判服务器上的运行时间超过了该编程任务已设定的最长时限。通常程序的循环次数太多，导致运行时间太长，需要改进算法。
memory Limit Exceed	MLE	内存超限	提交的程序在裁判服务器上运行所占用的内存空间超过了该编程任务已设定的最大限制。通常是数组太大或者累计未释放的动态分配空间太多。
Output Limit Exceed	OLE	输出超限	提交的程序产生了太多的输出。一般是程序没有正确地终止，没完没了地输出结果。

10.3 C 语言语法备查简要

- (1) 一个可执行的 C 语言程序有且仅有一个 `main()` 函数。
- (2) 变量必须先定义后使用。
- (3) 变量不能重复定义。
- (4) 变量在定义时并不一定能自动被初始化为 0，因此在第一次读取变量时，确保变量已经有正确的初值。
- (5) 同类型的多个变量可合并定义，中间用逗号分隔。
- (6) 所有标识符（包括常量名、变量名、函数名、数组名、指针名、结构体名、宏名等）必须遵守标识符命名规则。
- (7) 所有标识符区分大小写。
- (8) C 语言是格式自由的。只要不改变程序的语义，任何格式都是允许的。建议代码采用合适缩进风格，确保代码有较好可读性。
- (9) 诸如 `char, short, int, long long, float, double` 等 C 语言基本的内置数据类型。它们均有特定的存储方式、所占字节数、所能表示的数据范围和适用场合。计算机无法存储无穷大、无穷小的数据。现实世界中千变万化、千奇百怪的数据都是对这些基本数据类型的组合、编码、变换来得到。
- (10) 不同类型之间的数据进行赋值时，只有赋值兼容的类型之间才能进行直接赋值或实参到形参的参数传递。
- (11) C 语言支持强制类型转换，但是转换后数据的正确性由你自己负责。
- (12) C 语言数学运算符有 8 个：+（加）、-（减）、*（乘）、/（除）、%（取余）、-（反号）、++（自加）、--（自减）。
- (13) C 语言逻辑运算符有 3 个：&&（与）、||（或）、！（非）。
- (14) C 语言关系运算符有 6 个：>、>=、<、<=、==、!=。
- (15) C 语言位运算符有 6 个：&（按位与）、|（按位或）、~（按位非）、^（按位异或）、<<（左移）、>>（右移）。
- (16) 分支语句有 5 种：if、if-else、if-else if-else、switch-case。
- (17) C 语言有唯一的三目运算符：?:。它等价于一个 if-else 语句。
- (18) C 语言的循环语句只要 3 种：for、while、do-while。
- (19) C 语言的基本表达式有：算术表达式、关系表达式、逻辑表达式、位运算表达式、赋值表达式、逗号表达式。每个表达式均有值。
- (20) 赋值表达式的值即为被赋的值。
- (21) 逗号表达式的值为最右侧子表达式的值。即形如“`exp1, exp2, ..., expn`”逗号表达式，那么，逗号表达式的值为 `expn` 的值。
- (22) 赋值运算符只能给变量赋值，不能向数组名或常量赋值。
- (23) C 语言的所有运算符有固有的优先级和结合方向。
- (24) 只能使用左右小括号来改变运算的优先顺序，不能使用中括号和大括号来改变运算优先顺序。
- (25) 完整语句之后有分号。

- (26) 对于 `if,else if,while` 条件表达式的左右小括号必不可少。
- (27) 函数参数列表的左右小括号必不可少。
- (28) `switch` 表达式的左右小括号必不可少。
- (29) 当分支或循环执行的不是单个语句,而是多个语句,那么多个语句构成语句块,语句块的左右大括号必不可少。
- (30) 定义结构体类型的左右大括号必不可少。
- (31) 定义函数的函数体左右大括号必不可少。
- (32) `for` 语句头部分必须有一对左右小括号和 2 个分号。
- (33) 结构体定义的右大括号后必须有分号。
- (34) 绝非每行的末尾必有分号。
- (35) 循环头和循环体之间不能打分号。
- (36) 分支条件与分支语句(或语句块)之间不能打分号。
- (37) 函数头与函数体之间不能打分号。
- (38) 小括号、中括号、大括号、双引号、单引号必须正确配对。
- (39) 程序中使用库函数时,必须在程序开始处`#include` 库函数所在头文件。
- (40) 逻辑表达式的结果“逻辑真”为 1、“逻辑假”的值为 0。
- (41) 条件表达式的值为非 0 表示“条件成立”,条件表达式为 0 表示“条件不成立”。
- (42) 判断两个值是否相等的运算符是双等号“`==`”,不能用表示赋值运算符的单等号“`=`”。
- (43) 循环体中的 `break` 语句终止本层循环,并且只能跳出到本层循环。也就是说,在多层循环中,最内层的循环体中的 `break` 语句不能一次性跳出多层循环。如果程序逻辑需要实现一次性跳出多层循环,应该通过其它手段达到。
- (44) `continue` 语句将跳过当前位置之后本次循环体中的语句,直接进入下一次循环。
- (45) `if` 中的复合条件的表达必须通过多个单条件的“与`&&`,或`||`,!非”连接。
- (46) `else` 总是与所在代码块中最近的 `if` 配对。
- (47) `else` 必须与 `if` 配对。
- (48) 指向同类型数据的多个指针变量的定义必须在每个指针变量名前有“`*`”号。
- (49) 定义数组时,必须确定其大小。
- (50) 分配给数组的存储空间是连续的存储空间。
- (51) 数组的下标固定从 0 开始。
- (52) 通过“数组名[下标]”访问对应的数组元素。
- (53) 变量和数组均可在定义时初始化。
- (54) `printf()`、`scanf()`、`sprintf()`、`sscanf()` 中的%开头的格式控制符必须与输入/输出的变量一一对应。
- (55) 自定义函数的声明处必须位于调用处之前。
- (56) 函数声明时只有函数头,不能有函数体,即函数头后不能有表示函数体的`{ }`。
函数声明语句中的函数形式参数列表中可以省略形参名但是不能省略形参类型名。
- (57) 函数调用时实参和形参的个数、类型必须一一对应。
- (58) `return` 语句的作用是从当前位置结束本函数调用,回到主调函数。程序将从返回点处继续往后执行。

- (59) 函数返回值类型不为空的, 应该在适当位置用 `return` 语句返回适当值, 并且应该确保函数中的任何分支均有正确的返回值, 不能是有的分支有返回值, 有的分支没有返回值。
- (60) 1 个函数 1 次被调用后, 只能通过 `return` 语句带回 0 个或 1 个返回值。
- (61) 一个函数可以调用另一个函数。函数调用的显式的, 但是函数的返回时隐式。函数调用和返回的路径正确性依赖于“栈(stack)”这一特殊的数据结构。
- (62) 递归调用是函数自己调用自己。它必须能有正确的递归终点, 否则会“爆栈”----超过函数调用栈最大容量。
- (63) 字符数据 (不管是英文字符还是中文) 均是存储和处理其编码。
- (64) 英文字符的编码为 ASCII 码, 中文字符的编码为 GBK 码。
- (65) ASCII 码为 0 的字符, 即 ASCII 码表第一个字符, 通常用来表示字符串的结尾。
- (66) C 语言没有“字符串”类型, 因此字符串的存储和处理是以字符数组的方式实现的。字符串相关库函数对所处理的字符串均要求字符串末尾为空字符。
- (67) 常量字符串必须有双引号括起来, 常量字符必须由单引号括起来, 反斜杠“\”开头的字符为转义字符 (参见附表)。
- (68) 全局变量、局部变量、静态变量的作用范围、生命期、适用场合各不相同。局部变量名与全局变量名相同时, 在局部变量所在代码块中局部变量可见而全局变量不可见。
- (69) 指向任何数据类型的指针其本身大小为 32bits (这是对 32 位的 C 语言和 32 位的操作系统而言)。指针变量中存放的所指数据的起始地址。该地址值所占存储空间大小总是 32bits。
- (70) 对指针而言, 可对指针强制转换它所指的数据类型为任意其它类型。
- (71) 调用 `malloc()` 等函数动态分配的内存空间, 在不再使用时, 应该调用 `free()` 函数释放相应的空间。
- (72) `void` 用于声明函数返回值类型时表示该函数没有返回值。`void` 用于定义指针所指数据类型时, 表示该指针所指数据类型“待定”, 在通过该指针间接访问所指数据类型时必须先确定它所指数据的实际类型。
- (73) 通过关键字 `struct` 能定义处有多个数据项组合的新的数据类型。
- (74) 通过“结构体类型变量名.成员名”的方式, 可以访问某个结构体的成员。程序中, 对待结构体成员就像对待与此成员同类型的变量一样。
- (75) 通过“指向结构体类型的指针名->成员名”方式, 可访问指针所指结构体的成员。
- (76) 通过“结构体数组名[下标].成员名”的方式, 可以访问某下标结构体数组元素的成员。
- (77) 操作文件必须先调用 `fopen()` 函数打开文件, 通过文件指针操作文件, 使用文件完毕后应该调用 `fclose()` 关闭文件。
- (78) 读文件读写必须用正确的文件打开模式和正确文件读写函数进行操作。
- (79) 对文件的每次读/写都会自动使其读写位置向文件尾方向移动到下个适当位置。
- (80) 对文件进行交替读写时, 读写或写读之间必须调用 `fseek()` 函数。
- (81) 位运算是参与运算的数据在二进制下而不是十进制下的按位运算。

10.4 C 语言的保留关键字

C 语言的 32 个关键字(保留值)C 语言的关键字共有 32 个，根据关键字的作用，可分其为数据类型关键字、控制语句关键字、存储类型关键字和其它关键字四类。

类别		序号	类型名	说明
数据类型 关键字 (12 个)		(1)	char	声明字符型变量或函数
		(2)	double	声明双精度变量或函数
		(3)	enum	声明枚举类型
		(4)	float	声明浮点型变量或函数
		(5)	int	声明整型变量或函数
		(6)	long	声明长整型变量或函数
		(7)	short	声明短整型变量或函数
		(8)	signed	声明有符号类型变量或函数
		(9)	struct	声明结构体变量或函数
		(10)	union	声明联合数据类型（极少使用）
		(11)	unsigned	声明无符号类型变量或函数
		(12)	void	声明函数无返回值或无参数、声明 void*指针
控制 语句 关键字 (12 个)	循环 语句	(1)	for	一种循环语句(可意会不可言传)
		(2)	do	循环语句的循环体
		(3)	while	循环语句的循环条件
		(4)	break	跳出当前循环
		(5)	continue	结束当前循环，开始下一轮循环
	条件 语句	(1)	if:	条件语句
		(2)	else	条件语句否定分支（与 if 连用）
		(3)	goto	无条件跳转语句
	开关 语句	(1)	switch	用于开关语句
		(2)	case	开关语句分支
		(3)	default	开关语句中的“其他”分支
	返回		return	被调函数返回
存储类型 关键字 (4 个)		(1)	auto	声明自动变量,一般不使用
		(2)	extern	声明变量是在其他文件中已定义
		(3)	register	声明积存器变量
		(4)	static	声明静态变量
其它 关键字 (4 个)		(1)	const	声明只读变量
		(2)	sizeof	计算数据类型长度
		(3)	typedef	用以给数据类型取别名（当然还有其他作用
		(4)	volatile	说明变量在程序执行中可被隐含地改变

10.5 运算符的优先级和结合方向

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[]	数组下标	数组名[常量表达式]	从左到右	
	()	圆括号	(表达式) 函数名(形参表)		
	.	成员选择(对象)	对象.成员名		
	->	成员选择(指针)	对象指针->成员名		
2	-	负号运算符	-表达式	从右到左	单目运算符
	(类型)	强制类型转换	(数据类型)表达式		
	++	自增运算符	++变量名/变量名++		
	--	自减运算符	--变量名/变量名--		
	*	取值运算符	*指针变量		
	&	取地址运算符	&变量名		
	!	逻辑非运算符	!表达式		
	~	按位取反运算符	~表达式		
sizeof	长度运算符	sizeof(表达式)			
3	/	除	表达式/表达式	从左到右	双目运算符
	*	乘	表达式*表达式		
	%	余数(取模)	整型表达式/整型表达式		
4	+	加	表达式+表达式	从左到右	
	-	减	表达式-表达式		
5	<<	左移	变量<<表达式	从左到右	
	>>	右移	变量>>表达式		
6	>	大于	表达式>表达式	从左到右	
	>=	大于等于	表达式>=表达式		
	<	小于	表达式<表达式		
	<=	小于等于	表达式<=表达式		
7	==	等于	表达式==表达式	从左到右	
	!=	不等于	表达式!=表达式		
8	&	按位与	表达式&表达式	从左到右	
9	^	按位异或	表达式^表达式	从左到右	
10		按位或	表达式 表达式	从左到右	
11	&&	逻辑与	表达式&&表达式	从左到右	
12		逻辑或	表达式 表达式	从左到右	
13	?:	条件运算符	表达式1? 表达式2:表达式3	从右到左	三目运算符
14	=	赋值运算符	变量=表达式	从右到左	双目运算符
	/=	除后赋值	变量/= 表达式		
	=	乘后赋值	变量= 表达式		
	%=	取模后赋值	变量%= 表达式		
	+=	加后赋值	变量+= 表达式		
	-=	减后赋值	变量-= 表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&=	按位与后赋值	变量&=表达式		
	^=	按位异或后赋值	变量^=表达式		
=	按位或后赋值	变量 =表达式			
15	,	逗号运算符	表达式,表达式,...	从左到右	多目, 从左向右运算

上表中，优先级值越小表示优先级越高，优先级相同的按结合方向确定先后顺序。

以上表格不须死记，但须记住以下两个特点：

(1) 记住最高和最低的两头：方圆括号和成员选择优先级最高，其次是单目运算符；逗号的优先级最低，赋值运算的优先级仅高于逗号。

(2) 记住大类优先级：算术运算 > 移位运算 > 关系运算 > 按位与、按位或和按位异或 > 逻辑与或

如果在程序中需要判断变量 *i* 的值是否为 1，如果写成 `if (i=1) { }`，将是非常隐蔽的错误！因为 `if` 条件部分是“永真”，不管 *i* 原来的值为多少，此条件永远成立。理由很简单，`i=1` 是赋值表达式，该赋值表达式的值为 1，而值 1 为非零值，非零值表示“逻辑真”。所以在条件语句中一定要注意不要将“判断是否相等”的关系运算符（双等于号）写成赋值运算符（单等于号），这是初学者最容易犯的错误。

下表列举典型的优先级运用不当造成的错误。

优先级问题	表达式	被误认为	实际结果
.的优先级高于*和->操作符	<code>*p.f</code>	<code>(*p).f</code>	<code>*(p.f)</code>
<code>[]</code> 高于*	<code>int * ap[]</code>	<code>int (*ap)[]</code>	<code>int *(ap[])</code>
函数() <code>></code> 高于*	<code>int *fp()</code>	<code>int (*fp)()</code>	<code>int *(fp())</code>
<code>==</code> 和 <code>!=</code> 高于位操作	<code>(val & mask != 0)</code>	<code>((val & mask)!= 0)</code>	<code>(val & (mask != 0))</code>
<code>==</code> 和 <code>!=</code> 高于赋值符	<code>c = getchar() != EOF</code>	<code>(c =getchar()) != EOF</code>	<code>c =(getchar()) != EOF)</code>
算术运算符高于位移运算符	<code>msb <<4+lsb</code>	<code>(msb<< 4)+lsb</code>	<code>msb << (4 +lsb)</code>
逗号运算符在所有运算符中优先级最低	<code>i=1,2</code>	<code>i=(1,2)</code>	<code>(i=1),2</code>

10.6 输出格式控制符说明

以下为printf(“格式控制符”,输出列表)中的格式控制符的详细说明。

说明（1）：格式控制字符串都是以%开头。说明（2）：格式控制字符串一般格式为“%[-][0][m[.n]][l/h]格式符”，其中[]表示可选项。

格式符	含义	说明	举例	结果
—	指定左对齐输出	默认为右对齐输出	printf("%-6d\n",123);	123
0	指定在前导空位补零	只对右对齐方式起作用，0补在左侧。	printf("%06d\n",123);	000123
m.n	指定输出域宽度及精度。若实际数据宽度>m,按实际宽度输出；若实际数据宽度<m,则补空	用于浮点型时，m表示包括小数点在内的输出宽度，n表示保留小数的位数。 用于字符串型时，m为输出宽度，n为截留字符的个数	printf("%06.3s\n","abcde"); printf("%06.3lf\n",3.14159);	000abc 03.142
l/h	修饰长/短数据类型	对长整型为ld,lx,lo,lu; 对double型为lf; 对短整型为hd,hx,ho,hu	32位操作系统下，C语言中的long型和int型都是32位，没有区别。	
格式说明符	d	有符号十进制整数	int a=-1; printf("%d",a);	-1
	u	无符号十进制整数	int a=-1;printf("%u",a);	4294967295
	o	无符号八进制整数	int a=-1;printf("%o",a);	3777777777
	x或X	无符号十六进制整数	int a=-1; printf("%X",a);	FFFFFFFF
	f	以小数形式输出浮点数 对于float型，格式符为%f，前7位为有效数字，小数6位。 对于double型，格式符为%lf，前16位为有效数字，小数6位。	float a=3.14; printf("%f",a);	3.140000
			double a=3.14; printf("%lf",a);	3.140000
	e	以指数形式输出实型数	double a=3.14;printf("%e",a);	3.140000e+000
	g	按e和f中格式较短的形式输出	double a=3.14;printf("%g",a);	3.14
	c	输出一个字符。	char c='a'; printf("%c",c);	a
	s	输出一个字符串	char str[]="abcd"; printf("%s",a);	abcd

注：64位整数的输入输出：在Visual C++环境下，类型名为__int64（双下划线开头），输入输出格式控制符为“%I64d”。在G++或者gcc环境下，64位整数类型名为long long，输入输出

格式为”%lld”。

典型错误：数据类型与格式控制符不匹配。

```
doubleval; //变量val定义为double，即双精度型。
scanf(“%f”,&val); //输入时与double型相对应的格式控制符应该是%f,而不是%lf。
printf(“%f”,val); //输出时也应该使用与double型对应的格式控制符%f。
```

10.7 C 语言转义字符

常用转义字符：ASCII码表中的某些字符在程序中需要使用时，必须使用“\”才能表示，这些字符就称为转义字符。

常用转义字符表

转义 字符	转义后 得到的字符	备注
<code>\0</code>	空字符	(NULL),什么都不做，ASCII码为0
<code>\b</code>	退格	向后退一格
<code>\f</code>	换页	对屏幕没有任何影响，但会影响打印机执行响应操作
<code>\n</code>	换行	光标定位到下行行首换行，只是换一行，不改变光标的横坐标
<code>\r</code>	回车	光标定位到本行行首，回车只是回到行首，不改变光标的纵坐标
<code>\t</code>	水平制表	格式控制，跳格数由编译器控制
<code>\v</code>	垂直制表	对屏幕没有任何影响，但会影响打印机执行响应操作
<code>\\</code>	反斜杠	因为\本身作为转义符的开始字符，因此当在常量串中需要得到反斜杠字符本身时，需要用2个反斜杠。这在文件路径的常量字符串中经常遇到。
<code>\'</code>	单引号	如果单引号在字符串中，则使用\或直接使用单引号均可。只有当需要用到单个单引号字符时，才必须用\。例如， <code>printf("A\");</code> 输出'A'，此时要输出的单引号在字符串中。 <code>printf("%c","\");</code> 输出'\'，此时要输出的单引号在单字符中。
<code>\"</code>	双引号	因为双引号作为了常量字符的范围界定符。
<code>\ddd</code>	三位八进制	用3位八进制表示一个ASCII码为ddd的字符
<code>\xhh</code>	二位十六进制	用2位十六进制表示一个ASCII码为hh的字符

10.8 常用库函数简介

(1) 输入输出函数, 头文件:stdio.h, 即standard input/output header。

函数名	函数原型	功能	说明
scanf	int scanf(char * format, args , ...)	从标准输入按 format 指定的格式输入数据到 args 指定的位置。	返回成功输入数据的个数, 遇到文件结束则返回 EOF, 出错则返回 0。函数名助记: scan formatted。
fscanf	int fscanf(FILE * fp , const char *format , ...)	从 fp 所指向的文件中按 format 指定的格式输入数据到 args 指定的位置。	返回成功输入数据的个数, 遇到文件结束则返回 EOF, 出错则返回 0。函数名助记: file scan formatted。
sscanf	int sscanf(const char * str , const char * format , ...)	从字符串组 str 中按 format 指定的格式输入数据到 args 指定的位置。	返回成功输入数据的个数, 遇到文件结束则返回 EOF, 出错则返回 0。函数名助记: string scan formatted。
printf	int printf(char * format, args, ...)	按 format 的格式将 args 的值输出到标准输出设备。	返回输出字符的个数, 若出错则返回负数。函数名助记: 。 print formatted
fprintf	int fprintf(FILE * fp , const char * format , ...)	按 format 的格式将 args 的值输出到 fp 所指向的文件。	返回输出字符的个数, 若出错则返回负数。函数名助记: file print formatted。
sprintf	int sprintf(char * str, const char *format, ...)	按 format 的格式将 args 的值输出到字符串组 str。	返回输出字符的个数, 若出错则返回负数。函数名助记: string print formatted。
fgetc	int fgetc(FILE * fp)	从 fp 所指向的文件中读入一个字符。	返回所读字符, 若文件结束或出错则返回 EOF。函数名助记: file get character。
fputc	int fputc(int ch, FILE * fp)	向 fp 所指向的文件输出一个字符。	成功则返回字符 ch, 否则返回 EOF。函数名助记: file put character。
gets gets_s	char *gets(char *str) char *gets_s(char *str, rsize_t n)	从标准输入中读取	C11 标准之后采用 gets_s 取代 gets。函数名助记: get string。
fgets	char *fgets(char *str, int n, FILE *fp)	从 fp 指向的文件流中读入最长 n-1 个字符的字符串存放到字符串数组 str 中。	成功则返回 str, 否则返回 NULL。结果字符串 str 中将以'\0'结尾。如果遇到文件尾或字符个数到达 n-1 则立即返回。如果遇到换行符, 则将换行符也读入到 str, 然后返回。函数名助记: file get string。
puts	int puts(char *str)	将 str 中的字符串输出到标准输出设备, 遇到 str 末尾的'\0'输出回车。	成功则返回非负整数, 否则返回 EOF。函数名助记: put string。
fputs	int fputs(char * str, FILE * fp)	输出 str 中以'\0'结尾的字符串到 fp 指向的文件中。	成功则返回非负整数, 否则返回 EOF。函数名助记: file put string。
getchar	int getchar()	从标准输入设备中读取一个字符。	返回所读字符, 否则返回-1。
putchar	int putchar(int ch)	把字符 ch 输出到标注输出设备。	返回输出的字符 ch, 失败则返回 EOF。
fopen	FILE * fopen(char * filename, char * mode)	以 mode 指定的方式打开文件 filename。	成功则返回文件 filename 关联的文件指针, 否则返回 0。函数名助记: file open。

freopen	FILE *freopen(char * filename, char * mode, FILE * fp)	以 mode 指定的模式，将文件 filename 的输入和输出重定向到 fp 所指文件。	当模式为“读”时，则要从 fp 所指文件读取数据重定向为从 filename 文件读取数据；当模式为“写”时，则将写入数据到 fp 所指文件重定向为写入数据到 filename 文件；函数名助记：file reopen。 经典用法：重定向标准输入输出流到文件。
fclose	int fclose(FILE * fp)	关闭 fp 所指文件	成功则返回 0，否则返回非 0。函数名助记：file close。
fread	int fread(char *buf, unsigned size, unsigned n, FILE * fp)	从 fp 所指文件读取 n 块长度为 size 字节的数据到 buf	返回所读数据个数，如遇文件结果或出错则返回 0。函数名助记：file read。
fwrite	int fwrite(char *buf, unsigned size, unsigned n, FILE * fp)	将 buf 中 n 块长度为 size 字节的数据写入到 fp 所指文件	返回写入到文件中的数据块的个数。函数名助记：file write。
ftell	long ftell(FILE * fp)	返回 fp 所指文件的读写指示器的当前位置。	返回文件读写位置指示器的当前位置相对文件头的偏移字节数，失败则返回 EOF。函数名助记：file tell。
rewind	void rewind(FILE * fp)	将 fp 所指文件的当前读写位置拨回到文件开始处。	注意此函数名不要误作 frewind。
fseek	int fseek(FILE * fp, long offset, int base)	以 offset 为偏移量，base 为基准，拨动 fp 所指文件的读写位置指示器到指定位置。	返回文件读写位置指示器的当前位置，失败则返回-1。
feof	int feof(FILE * fp)	检查文件是否结束。	遇到文件结束符则返回非 0，否则返回 0。
fflush	int fflush(FILE * stream)	使输出流的缓存数据写入到实际文件。	成功则返回 0，否则返回 EOF，并设置文件流错误信息。函数名助记：file flush。
rename	int rename(char * oldname, char * newname)	将文件 oldname 改名为 newname。	成功则返回 0，否则返回-1。
remove	int remove(char * filename)	删除文件名为 filename 的文件	成功则返回 0，否则非 0。
tmpfile	FILE * tmpfile()	以"wb+"模式打开一个临时文件，并保证此临时文件是唯一的。	函数名助记：temporary file。

(2) 数学函数, 头文件为 `math.h`

函数名	函数原型	功能	说明
<code>abs</code>	<code>int abs(int x)</code>	求整数 x 的绝对值	返回值类型是 <code>int</code> ; 函数名助记: absolute value, 绝对值。
<code>fabs</code>	<code>double fabs(double x)</code>	求实型数 x 的绝对值	返回值类型是 <code>double</code> ; 函数名助记: float absolute value, 浮点数绝对值。
<code>exp</code>	<code>double exp(double x)</code>	求 e^x 的值	e 为自然常数, 注意结果值不能超过 <code>double</code> 的表示范围; 函数名助记: exponent, 指数。
<code>pow</code>	<code>double pow(double x, double y)</code>	求 x^y 的值	注意结果值不能超过 <code>double</code> 的表示范围; 函数名助记: power, 幂。
<code>log</code>	<code>double log(double x)</code>	求 $\log_e x$, 自然对数	x 应大于 0; 函数名助记: logarithm, 对数
<code>log10</code>	<code>double log10(double x)</code>	求 $\log_{10} x$ 的值	x 应大于 0
<code>log2</code>	<code>double log2(double x)</code>	求 $\log_2 x$ 的值	x 应大于 0
<code>floor</code>	<code>double floor(double x)</code>	求不大于 x 的最大整数	$\lfloor x \rfloor$, 向下取整, 返回值是 <code>double</code> 型, 不是整型。 例: <code>floor(5.2)=5.0</code> , <code>floor(-5.6)=-6.0</code>
<code>ceil</code>	<code>double ceil(double x)</code>	求不小于 x 的最小整数	$\lceil x \rceil$, 向上取整, 返回值是 <code>double</code> 型, 不是整型。 <code>ceil(5.2)=6.0</code> , <code>ceil(-5.6)=-5.0</code>
<code>round</code>	<code>double round(double x)</code>	对 x 四舍五入	C99 标准中引入。 例: <code>round(1.1)=1.0</code> <code>round(1.5)=2.0</code> <code>round(-1.1)=-1.0</code> <code>round(-1.5)=-2.0</code>
<code>trunc</code>	<code>double trunc(double x)</code>	截去 x 的小数部分	C99 标准中引入。 例: <code>trunc(1.1)=1.0</code> <code>trunc(1.5)=1.0</code> <code>trunc(-1.1)=-1.0</code> <code>trunc(-1.5)=-1.0</code>
<code>fmod</code>	<code>double fmod(double x, double y)</code>	求 x/y 整除的余数	例: <code>fmod(6.4,3.1)=0.2</code> 函数名助记: float modulo, 浮点数的模运算
<code>sqrt</code>	<code>double sqrt(double x)</code>	求 \sqrt{x} 的值	x 应大于等于 0; 函数名助记: square root, 平方根。
<code>cbrt</code>	<code>double cbrt(double x)</code>	求 x 的立方根	x 应大于等于 0; C99 标准中引入; 函数名助记: cubic root, 立方根;
<code>sin</code>	<code>double sin(double x)</code>	求 x 的正弦值	x 的单位为弧度。函数名助记: sine
<code>cos</code>	<code>double cos(double x)</code>	求 x 的余弦值	x 的单位为弧度。函数名助记: cosine
<code>tan</code>	<code>double tan(double x)</code>	求 x 的正切值	x 的单位为弧度。函数名助记: tangent
<code>asin</code>	<code>double asin(double x)</code>	求 x 的反正弦值	x 应在 $[-1,1]$ 内取值。函数名助记: arc sine
<code>acos</code>	<code>double acos(double x)</code>	求 x 的反余弦值	x 应在 $[-1,1]$ 内取值。函数名助记: arc cosine
<code>atan</code>	<code>double atan(double x)</code>	求 x 的反正切值	函数名助记: arc tangent
<code>sinh</code>	<code>double sinh(double x)</code>	求 x 的双曲正弦值	函数名助记: hyperbolic sine
<code>cosh</code>	<code>double cosh(double x)</code>	求 x 的双曲余弦值	函数名助记: hyperbolic cos
<code>tanh</code>	<code>double tanh(double x)</code>	求 x 的双曲正切值	函数名助记: hyperbolic tangent
<code>asinh</code>	<code>double asinh(double x)</code>	求 x 的反双曲正弦值	函数名助记: hyperbolic arc sine
<code>acosh</code>	<code>double acosh(double x)</code>	求 x 的反双曲余弦值	函数名助记: hyperbolic arc cosine
<code>atanh</code>	<code>double atanh(double x)</code>	求 x 的反双曲正切值	函数名助记: hyperbolic arc tangent

(3) 字符串操作函数, 头文件: string.h

请注意: 以下字符串函数中, 要求每个参与运算的字符串都是以'\0'结尾。

函数名	函数原型	功能	说明
strcat	char * strcat(char * str1, char * str2)	把字符串 str2 拼接到 str1 之后	返回 str1, 必须保证 str1 有足够的空间存放拼接后字符串。 函数名助记: string concatenate
strcmp	char * strcmp(char * str1, char * str2)	比较两个字符串 str1, str2	当 str1 小于 str2 时, 返回值<0 当 str1 等于 str2 时, 返回值==0 当 str1 大于 str2 时, 返回值>0 函数名助记: string compare
strcpy	char * strcpy(char * str, char * str2)	字符串 str2 复制到 str1	返回 str1, 必须保证 str1 有足够的空间存放拼接后字符串。 函数名助记: string copy
strlen	unsigned int strlen(char * str)	统计字符串 str 中字符的个数 (不包括'\0')	返回字符串 str 的长度, 即字符个数。 函数名助记: string length
strchr	char * strchr(char * str, char ch)	查找字符 ch 在字符串 str 中第一次出现的位置	单字符的查找。 返回指向第一次出现位置的指针, 找不到则返回空指针。函数名助记: string character。
strpbrk	char * strpbrk(char * str1, char * str2)	查找 str2 中的字符在 str1 中最早第一次出现的位置	单字符查找。 返回指向第一次出现位置的指针, 找不到则返回空指针。 char s1[]="hello", s2="oh" strpbrk(s1,s2)返回的指针指向字符'h'处。 函数名助记: string break。
strstr	char * strstr(char str1, char str2)	查找字符串 str2 在字符串 str1 中第一次出现的位置。	子串的查找。 返回指向第一次出现位置的指针, 找不到则返回空指针。 函数名助记: string in string。
strlwr	char * strlwr(char * str)	将 str 中的大写字母转换为小写字母。	返回 str 函数名助记: string lower
strupr	char *strupr(char * str)	将 str 中的小写字母转换为大写字母。	返回 str 函数名助记: string upper
memset	void* memset(void* dest, int ch, size_t count)	将 dest 开始的 count 个字节的内存按字节赋值为 ch	按字节给内存块赋值; 返回 dest。 按字节赋值前先把 ch 转换为 unsigned char, 再将此值按字节赋给 dest 开始的连续 count 个字节的存储区。 函数名助记: memory set
memcpy memmove	void* memcpy(void* dest, const void* src, size_t count) void* memmove(void* dest, const void* src, size_t count)	将 src 开始的 count 字节复制到 dest	按字节拷贝内存块; 返回 dest; 函数名助记: memory copy, memory move。 应防止数据源空间与目标空间重叠。
memcmp	int memcmp(void* left, const void* right, size_t count)	按字节比较 left 和 right 中的 count 字节。	如果 left 小于 right, 返回值<0 如果 left 等于 right, 返回值==0 如果 left 大于 right, 返回值>0 函数名助记: memory compare。

(4) 标准库函数, 头文件: `stdlib.h` (standard library header), 包括数值转换函数、动态存储分配函数。

函数名	函数原型	功能	说明
<code>atoi</code>	<code>int atoi(char * str)</code>	将字符串 <code>str</code> 转换为 <code>int</code> 型整数	返回转换后的 <code>int</code> 型整数, 失败返回 0。函数名助记: ASCII to integer。
<code>atol</code>	<code>long atoi(char * str)</code>	将字符串 <code>str</code> 转换为 <code>long</code> 型整数	返回转换后的 <code>long</code> 型整数, 失败返回 0。函数名助记: ASCII to long。
<code>atoll</code>	<code>long long atoi(char * str)</code>	将字符串 <code>str</code> 转换为 <code>long long</code> 型整数	返回转换后的 <code>long long</code> 型整数, 失败返回 0。函数名助记: ASCII to long long。
<code>atof</code>	<code>double atof(char * str)</code>	将字符串 <code>str</code> 转换为双精度浮点数	返回转换后的双精度浮点数, 失败返回 0。函数名助记: ASCII to float。
<code>strtof</code>	<code>float strtof(char * str, char ** str_end)</code>	转换字符串 <code>str</code> 为单精度浮点数	忽略前导空白字符, 返回转换后的 <code>float</code> 值, 不能转换则返回 0, 第二个参数一般设置为 <code>NULL</code> 。C99 标准。函数名助记: string to float。
<code>strtod</code>	<code>double strtod(char * str, char ** str_end)</code>	转换字符串 <code>str</code> 为双精度浮点数	忽略前导空白字符, 返回转换后的 <code>double</code> 值, 不能转换则返回 0, 第二个参数一般设置为 <code>NULL</code> 。函数名助记: string to double。
<code>strtol</code>	<code>long strtol(char * str, char **str_end,int base)</code>	将字符串 <code>str</code> 中的数字作为 <code>base</code> 进制的 <code>long</code> 型整数。	忽略前导空白字符, 返回转换后的 <code>long</code> 值, 不能转换则返回 0, 第二个参数一般设置为 <code>NULL</code> 。函数名助记: string to long。
<code>strtoll</code>	<code>long long strtol(char * str, char **str_end,int base)</code>	将字符串 <code>str</code> 中的数字作为 <code>base</code> 进制的 <code>long long</code> 型整数。	忽略前导空白字符, 返回转换后的 <code>long long</code> 值, 不能转换则返回 0, 第二个参数一般设置为 <code>NULL</code> 。C99 标准。函数名助记: string to long long。
<code>strtoul</code>	<code>unsigned long strtoul(char * str, char **str_end,int base)</code>	将字符串 <code>str</code> 中的数字作为 <code>base</code> 进制的 <code>unsigned long</code> 型整数。	忽略前导空白字符, 返回转换后的 <code>unsigned long</code> 值, 不能转换则返回 0, 第二个参数一般设置为 <code>NULL</code> 。函数名助记: string to unsigned long。
<code>strtoull</code>	<code>unsigned long long strtoll(char * str, char **str_end,int base)</code>	将字符串 <code>str</code> 中的数字作为 <code>base</code> 进制的 <code>unsigned long long</code> 型整数。	忽略前导空白字符, 返回转换后的 <code>unsigned long long</code> 值, 不能转换则返回 0, 第二个参数一般设置为 <code>NULL</code> 。C99 标准。函数名助记: string to unsigned long long。
<code>rand</code>	<code>double rand()</code>	产生[0,32767]的随机整数	函数名助记: random,随机;
<code>srand</code>	<code>void srand(unsigned seed)</code>	用随机种子 <code>seed</code> 初始化随机数发生器	和 <code>rand()</code> 函数配合使用; 函数名助记 <code>seed random</code> , 随机种子。
<code>calloc</code>	<code>void * calloc(unsigned n, unsigned size)</code>	分配 <code>n</code> 个每块为 <code>size</code> 字节的动态内存连续空间。被分配空间按字节清空位 0。	分配得到的内存块起始地址。不成功则返回 <code>NULL</code> 。函数名助记: clear allocation。
<code>malloc</code>	<code>void * malloc(unsigned size)</code>	分配 <code>size</code> 字节的动态内存连续空间。	分配得到内存块的起始地址。不成功则返回 <code>NULL</code> 。函数名助记: memory allocation。
<code>realloc</code>	<code>void * realloc(void * buf, unsigned size)</code>	将 <code>buf</code> 指向已分配动态内存空间的大小改为 <code>size</code> 。 <code>size</code> 可比原空间大或小。	返回调整后的内存块起始地址。原内存空间中的数据将复制到新内存空间。函数名助记: reallocation。

free	void free(void * p)	释放 p 指针指向的已分配的动态内存空间。	
abort	void abort()	使程序非正常终止，其不做清除操作。	
exit	void exit(int exit_code)	使程序正常终止，执行清除操作。	返回码 exit_code 为 EXIT_SUCCESS，表示退出状态为“成功”，EXIT_FAILURE，表示退出状态为“不成功”。清除操作包括：调用传递给 atexit()的函数；保存全部输入输出流，并关闭输入输出流；清楚所有由 tmpfile()创建的临时文件；将控制返回给操作系统。
_Exit	void _Exit(int exit_code)	使程序正常终止，但不执行清除操作。	返回码 exit_code 为 EXIT_SUCCESS，表示退出状态为“成功”，EXIT_FAILURE，表示退出状态为“不成功”。C99 标准。
atexit	int atexit(void(* func)())	注册执行 exit()退出时将被调用的函数 func。	注册成功返回 0，否则返回非 0。多个注册函数的调用顺序与注册顺序相反。
quick_exit	void quick_exit(int exit_code)	使程序正常终止，不执行清除操作，但将输入输出缓冲与文件同步。	传递给 at_quick_exit 的函数将被调用，然后调用 _Exit (exit_code)。C11 标准。
at_quick_exit	int at_quick_exit(void(* func)())	注册执行 quick_exit()退出时将被调用的函数 func。	注册成功返回 0，否则返回非 0。多个注册函数的调用顺序与注册顺序相反。
getenv	char *getenv(char *env_var)	查找名为 env_var 的环境变量的值。	返回环境变量的值，找不到则返回 NULL。函数名助记：get environment variable。
system	int system(char * command)	调用命令行程序执行 command 参数指定的命令。	返回被调用程序的返回值。

(5) 字符判别和转换函数，头文件：ctype.h

函数名	函数原型	功能	说明
isdigit	int isdigit(int ch)	检查字符 ch 是否都为数字 0~9	是则返回 1，否则返回 0；
isalpha	int isalpha(int ch)	检查字符 ch 是否为字母	是则返回 1，否则返回 0； 函数名助记：is alphabet
isalnum	int isalnum(int ch)	检查字符 ch 是否为字母或数字	是则返回 1，否则返回 0； 函数名助记：is alphabet or number
isxdigit	int isxdigit(int ch)	检查 ch 是否为十六进制字符（0~9, a~f, A~F。	是则返回 1，否则返回 0；函数名助记：is hexadecimal digit。
ispunct	int ispunct(int ch)	检查是否为标点字符（除空格、字母、数字之外的可打印字符）	是则返回 1，否则返回 0； 函数名助记：is punctuation
isspace	int isspace(int ch)	检查 ch 是否为空格(0x20)、换页(0x0c)、换行(0x0a)、回车(0x0d)、水平跳格(0x09)、垂直跳格(0x0b)	是则返回 1，否则返回 0；
isblank	int isblank(int ch)	检查 ch 是否为跳格或空格	是则返回 1，否则返回 0；
isgraph	int isgraph(int ch)	检查 ch 是否为可打印字符（ASCII 码为 0x21~0x7e），不包括空格	是则返回 1，否则返回 0；
isprint	int isprint(int ch)	检查 ch 是否为可打印字符（ASCII 码为 0x20~0x7e），包括空格	是则返回 1，否则返回 0；
isctrl	int isctrl(int ch)	检查 ch 是否为控制字符（ASCII 码为 0x00~0x1f）	是则返回 1，否则返回 0； 函数名助记：is control
islower	int islower(int ch)	检查 ch 是否为小写字母 a~z	是则返回 1，否则返回 0；
isupper	int isupper(int ch)	检查 ch 是否为大写字母 A~Z	是则返回 1，否则返回 0；
tolower	int tolower(int ch)	将 ch 转换为小写字母	返回 ch 的小写字母
toupper	int toupper(int ch)	将 ch 转换为大写字母	返回 ch 的大写字母

(6) 时间日期函数, 头文件: `time.h`

函数名	函数原型	功能	说明
<code>difftime</code>	<code>double difftime(time_t tm2, time_t tm1)</code>	计算 <code>tm2</code> 与 <code>tm1</code> 之间的时间差, 以秒为单位。	函数名助记: 。 difference of time
<code>time</code>	<code>time_t time(time_t * time_ptr)</code>	得到当前系统时间。	返回值当前时间, 并放在 <code>time_ptr</code> 指向的 <code>time_t</code> 结构体。 <code>tm</code> 可以为 <code>NULL</code> 。失败返回 <code>time_t(-1)</code> 。
<code>clock</code>	<code>clock_t clock()</code>	得到程序启动以来处理器的时钟滴答数。	无法获得时间则返回-1。一般需要两次 <code>clock()</code> 调用的差除以 <code>CLOCKS_PER_SEC</code> 。
<code>asctime</code>	<code>char * asctime(const tm* time_ptr)</code>	转换 <code>time_ptr</code> 指向的日历时间对象为文本形式。	结果字符串的形式如下: <code>Www Mmm dd hh:mm:ss yyyy</code> 。函数名助记: ASCII time。
<code>ctime</code>	<code>char * ctime(time_t * time)</code>	转换 <code>time</code> 指向的纪元时间为文本形式。	结果字符串的形式如下: <code>Www Mmm dd hh:mm:ss yyyy</code> 。函数名助记: epoch time。
<code>strftime</code>	<code>size_t strftime(char * str, size_t count, char * format, tm* time)</code>	转换 <code>tm</code> 对象为自定义文本形式。	返回结果字符串的长度。以 <code>format</code> 指定的格式, 转换 <code>tm</code> 指向的日历时间对象为 <code>str</code> 指向的字符串, 串长度最大为 <code>count</code> 。函数名助记: string format time。
<code>gmtime</code>	<code>tm* gmtime(time_t* time)</code>	转换纪元时间为 UTC 表示的日历时间	北京时区属于东八区, 北京时间领先 UTC 时间 8 小时。
<code>localtime</code>	<code>tm* localtime(time_t * time)</code>	转换纪元时间为 UTC 表示的当地日历时间。	
<code>mktime</code>	<code>time_t mktime(tm* time)</code>	转换 <code>time</code> 指向的日历时间为纪元时间。	返回纪元时间, 失败则返回-1。函数名助记: make time。
<code>CLOCK_PER_SEC</code>	常量	处理器每秒滴答数。	名字助记: clock per second
<code>tm</code>		日历时间结构体类型	成员: <code>int tm_sec</code> 秒, [0,60] <code>int tm_min</code> 分, [0,59] <code>int tm_hour</code> 时, [0,23] <code>int tm_day</code> 天, [1,31] <code>int tm_mon</code> 月, [0,11] <code>int tm_year</code> 自 1900 以来的年数 <code>int tm_wday</code> 星期几, [0,6] <code>int tm_yday</code> 自元月一日以来的天数, [0,365] <code>int tm_isdst</code> 夏令时标志, 正值表示为夏令时。名字助记: is daylight saving time。
<code>time_t</code>		纪元时间类型	表示自 1970 年元月一日 0 时 0 分以来的秒数 (整数), 与具体实现有关。
<code>clock_t</code>		处理器运行时间类型	能满足表示处理器运行时间的范围和精度的数值类型, 与具体实现有关。

更多 C 语言库函资料请参考:

- (1) C 语言函数参考手册, 明日科技, 清华大学出版社, 2012
- (2) 维基百科“C 标准库”, http://en.wikipedia.org/wiki/C_standard_library。
- (3) C++参考, <http://cplusplus.com>

10.9 编译前的预处理

10.9.1 宏定义#define

宏定义是 C 语言支持的预处理功能之一。它有 2 种形式的宏定义。

其一，不带参数的宏定义，形如：**#define 宏名 字符串**

<pre>#include <stdio.h> #define PI 3.14159265 int main(){ double r,s,S; scanf("%lf",&r); S=PI*r*r; s=2*PI*r; printf("%lf\n%lf",S,s); return 0; }</pre>	<p>此程序的功能是求给定半径的圆面积和圆周长。</p> <p>在此定义了宏 PI，在程序中，可以将它当作符号常量使用。</p> <p>在此 2 次引用了宏 PI，为程序带来易读性和易维护性。</p> <p>例如，如果需要修改 PI 的值，只需修改 1 处即可，即修改 PI 的宏定义处即可。如果代码中多处直接使用 π 数字常量，则需要修改多处，容易遗漏和出错。</p>
--	--

例如，**#define PI 3.14159**，那么在程序中使用 PI

其二，带参数的宏定义，形如：**#define 宏名(参数列表) 字符串**。这种形式下，“宏名、(、逗号分隔的参数列表、)”所构成的串中不能有空格。

<pre>#include <stdio.h> #define PI 3.14159265 #define AREA(r) PI*(r)*(r) #define LENS(r) 2*PI*(r) int main(){ double bj; scanf("%lf",&bj); printf("%lf\n",AREA(bj)); printf("%lf\n",LENS(bj)); return 0; }</pre>	<p>此程序的功能是求给定半径的圆面积和圆周长。</p> <p>在此定义了宏 PI，它是无参数的宏。</p> <p>在此定义了 2 个带参数的宏，参数为 r，在此请注意，串中的参数 r 均被一对小括号括起来了。此宏定义引用了另一宏 PI。</p> <p>这里引用两个带参数的宏。形式上看起来像函数的调用，其实质并非函数调用，而是预编译时的宏展开。除了宏总体被替换外，其中的参数 bj 将替换宏定义中参数 r。</p>
--	--

值得注意的是：带参数宏定义中，每个参数的每处出现务必要用一对小括号括起来，这对括号看似多此一举，实则不然。如果不这样作可能会有隐蔽错误。

举例如下：假如，上例中的 AREA(r) 宏定义为 “AREA(r) PI*r*r”，有两个变量 a, b 的值分别为 2, 8，那么程序中引用宏 AREA(a+b) 得到的期望结果应该是 314.159265，而运行后得到实际运行结果为 30.283185，这是为什么呢？原来，宏 AREA(a+b) 展开后是这样的：3.14159265*a+b*a+b，显然，这样展开后，优先级发生了改变，不是我们想要的结果了。

关于宏的补充说明：

(1) 程序中定义的宏，是在程序预编译阶段（即编译前的预处理阶段）替换成相应的字符串的，此过程称为“宏展开”。展开过程实质是字符串替换，不进行任何语法检查。要等到编译阶段才会进行语法检查。

(2) 宏定义预处理命令，不是一个 C 语句，通常情况下，定义宏的行尾不要有分号。如果有分号，在宏展开时，此分号将作为串的部分参与替换。

(3) 通常情况下，宏有 3 种用途。其一，不带参数的宏来将程序中的数值常量用符号常量（即宏名）表示，使程序代码的可读性增强。例如，我们处理某次考试成绩数据，正常值 [0, 100] 范围，成绩定义 -1 表示学生缺考。那么程序中出现 -1 这个值并不好理解，如果用符号常量 “QUE_KAO” 表示则可读性好很多。要做到这一点只需要在程序开始部分增加一行宏定义 “#define QUE_KAO 1” 即可。其二，带参数的宏，可以用来带有参数的复杂表达式用宏来表达。当然，这也可用带参数的函数来实现，但是宏的运行效率更高，因为宏在预编译阶段直接展开成表达式的，而不是像 “函数” 在运行时有参数传递、压调用栈、弹调用栈、返回值等复杂过程。其三，与条件编译配合，实现条件编译。

(4) 建议宏名使用大写字母，以示区别。但这并非强制规定。

(5) 一个宏的定义中可以引用另一个宏，即允许嵌套引用。但是，宏不能递归定义，即不能在宏定义中引用自己。

(6) 多次重复定义同一个宏，结果以最后一个宏为准。

(7) 宏的作用范围为从定义点开始至它所在代码文件尾。其作用范围只与其定义在代码中位置有关系。也就是说，宏的作用范围与其是定义在函数内还是函数外没有关系，即使宏定义是在函数内，其作用范围也与该函数是否被调用没有关系。这一点请读者自己验证。

(8) “#undef 宏名” 的方式提前结束某宏的作用范围。

(9) 形如 #define 宏名 也是允许的，此时只有宏名，没有后面的字符串部分。通常，此宏作为条件编译的条件使用。

(10) C 语言中有几个特殊的宏，它们都是双下划线开头和结尾。

宏名	数据类型	说明
__FILE__	字符串	此宏代码所在文件名（带全路径）
__FUNCTION__	字符串	此宏代码所在函数名
__LINE__	无符号整数	此宏代码在所在文件的行号
__DATE__	字符串，形如 Dec 31 2019	此宏所在文件的修改日期
__TIME__	字符串，形如 00:00:30	此宏所在文件的修改时间

(11) 因为宏展开是在预编译阶段进行字符串替换的本质，利用此特点，有各种脑洞大开的用法。例如下例仅仅是为了展示宏定义的 “字符串替换” 的本质，并不建议这样写程序。

```

#include <stdio.h>
#define mult(a,b) (a)*(b);
#define S "%d\n"
#define println printf("\n");
#define prtln(val) printf("%d\n",val);
#define prttab(val) printf("%d\t",val);
#define arr(i,j) arr[i][j]
#define P printf("%d\n"
#define CODE int x=2,y=8,z;\
                for(z=x,i=0;i<y;i++)\
                    z*=x;\
                printf("x的y次幂=%d\n",z);
int main(){
    int a=1,b=2,c=3,d=4,i,j;
    printf(S,mult(3,4))
    println(b*d)
    int arr(2,3)={1,4,7,3,6,9};
    for(i=0;i<2;i++){
        for(j=0;j<3;j++)
            prttab(arr(i,j))
        println
    }
    P,56);
    CODE
    return 0;
}

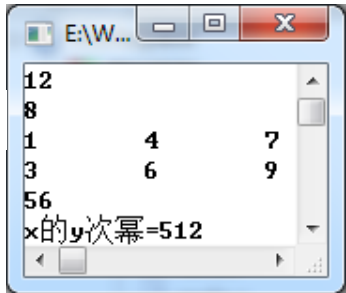
```

在此展示各种样子诡异的宏。但是本程序确实能正确运行。

请注意，因为宏定义的串是不能直接换行的。如果需要将本应在一行的代码写成多行，必须使用续行符\。反斜杠作为续行符只能在断行最末尾处。

这看似奇怪的代码，利用宏展开后，就不奇怪了。

运行结果



利用宏定义改变了二维数组元素的引用形式。

这些看似奇怪的代码，利用宏展开后，就不奇怪了。

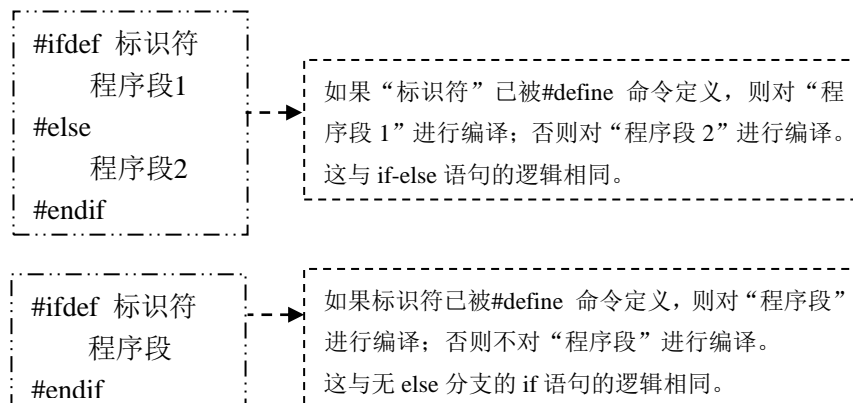
实质上，不管是有参和无参的宏定义，均是在编译前的预处理阶段，按“字符串替换”的方式处理你的源代码，然后才进行编译的。

10.9.2 条件编译

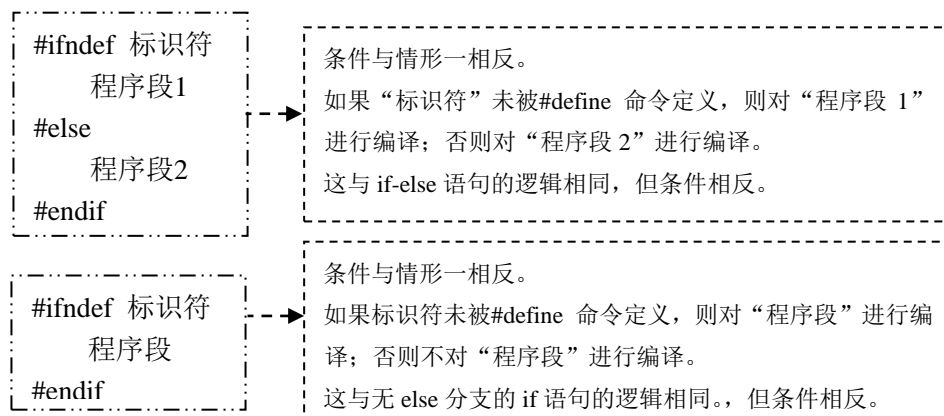
C语言支持条件编译，这使得我们的程序在编译时，只要在某处改变一下条件，编译器就能根据不同的条件编译不同的程序段，从而得到不同的目标代码文件。这对于程序的调试和的移植非常有用。

条件编译有三种基本形式，与if-else语句一样，允许嵌套。

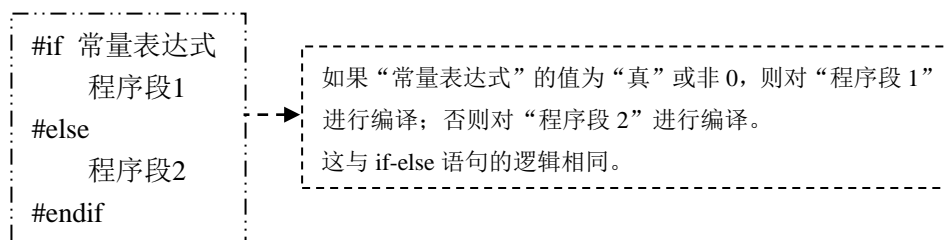
基本形式一：



基本形式二：



基本形式三：



在软件工程中的软件项目，通常会有debug版和release版，分别表示调试版和发行版。debug版的程序功能与release版相同的，其实release版就是不输出调试信息的debug版。调试版是为了方便软件的维护与升级而设置的，因此，debug版中设置了许多为了方便程序调试和观察而输出信息的大量语句，这些语句需要在生成软件发行版时通通不起作用。此时，利用条件编译可以很好地达到此目的。

Code::Block编程环境已经为你定义了宏DEBUG，当你的代码只需要在debug版中起作用时，只需将代码放在“#ifdef DEBUG”与“#endif”之间即可。如下代码所示。

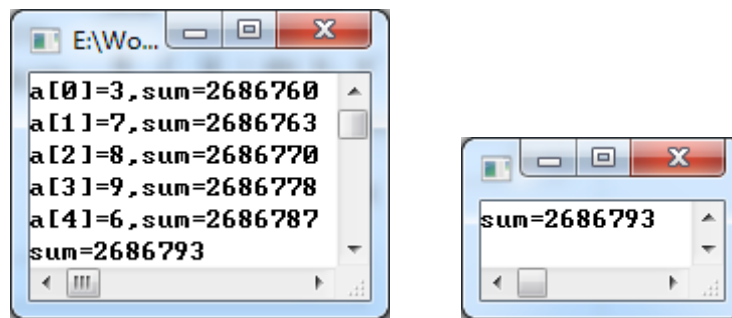
```

#include <stdio.h>
int main(){
    int sum,i, a[5]={3,7,8,9,6};
    for(i=0;i<5;i++){
        #ifdef DEBUG
            printf("a[%d]=%d,sum=%d\n",
                i,a[i],sum);
        #endif
        sum=sum + a[i];
    }
    printf("sum=%d \n",sum);
    return 0;
}

```

在此，设置条件编译的宏，此条件表示当程序中，有`#define DEBUG`，这意味着当前的项目是 debug 版，那么程序将编译`#ifdef` 与`#endif` 之间的代码，其效果是输出每个数组元素的值和每次累加后 `sum` 的值，这是为观察累加的中间过程是否正确而输出的调试信息。

如果我们只要选择Code::Blocks工程的菜单“Build Select Target debug或release”菜单项，就能看到输出了调试信息和不输出调试信息两种结果。或者是我们不去切换当前工程的debug版和release版，而是直接将此代码的`#ifdef`改为`#ifndef`，也能看到输出了调试信息和不输出调试信息两种结果。如下图所示。



输出了调试信息的结果

没有输出调试信息的结果

通过观察输出的调试信息，就能很快发现，累加和结果错误的原因是`sum`变量在累加前没有赋初值。

10.9.3 文件包含

文件包含的重要意义有两点，其一是减少重复劳动。例如，将程序中符号常量集中到一个文件，其它代码文件只需要`#include`这个文件即可。再例如，利用库函数，只需要`include`相应头文件即可。其二，使得大源代码文件能够分成多个多个小的源代码文件，换个角度来说，使得一个软件项目能由多个.c和.h文件构成，便于管理和维护。

在我们编写的第一个C语言程序就已经使用“文件包含”了。文件包含是C语言预处理的一个重要功能。文件包含是指，在预编译阶段，一个源文件将另一个源文件包含进来，然后再一起编译。C语言提供`#include`命令来实现文件包含的操作，它实际是宏替换的延伸，有两种格式。

格式1: `#include <filename>`

其中，`filename` 为要包含的文件名称，也称为头文件。此命令意味着，从C语言编译环境的库函数头文件所在目录搜索此文件。找到文件后，用文件内容替换该语句。

格式2: `#include "filename"`

其中，`filename`为要包含的文件名称。与上述尖括号方式的唯一区别在于：首先从本程序项目的当前目录中查找`filename`文件，若没有找到，则再按上述方式搜索。

`include` 支持相对路径，其中.（单点号）代表当前目录，..（双点号）代表上层目录。

10.10 程序调试

程序设计时经常会遇到这种情况，程序能通过编译，但是程序的运行结果总是不正确，或者程序对于某些测试用例的输出不正确，此时应该怎么办？

解决问题的步骤大致有如下：

首先，审查该程序所用算法是否正确？如果算法不正确，则应该修改算法，代码当然也要相应作修改。如果算法没有问题，应该是程序的问题，那么需要对程序进行调试。

在调试程序前，必须找到一组测试数据，该测试数据的正确结果已知，但是程序运行得到的结果不正确。

调试程序时总的做法是：针对特定的测试数据，将它代入你的程序，观察程序的每一步是否按照你预定算法思路的路径在执行。

当然，如果，要做到观察每一步执行有困难，那么必须做到重点观察得到中间结果的那些重要步骤。

为了方便观察，可以具体来说，有两种做法，其一，” debug+断点”法，其二，输出中间变量法。

“debug+断点”法：通常，C语言编程环境支持以 debug 模式运行程序，在这种模式下，程序将在你设置了断点的语句暂停下来，方便我们对程序的当前状态进行观察。这主要包括：观察程序当前输出结果、查看指定变量的值、查看指定内存空间的值、查看当前调用堆栈情况等。观察完毕后，如果一切正常，则有多种方式继续进行程序，通常包含：设置或取消断

点、单步执行、执行到光标所在位置、进入函数、跳出函数等。直到发现问题代码所在位置为止。然后，退出调试状态，修改程序，进入下一个测试和调试周期。直到程序正确。这种方法的优点能发现隐蔽较深、非常细微的错误。缺点是耗时、费力，通常需要对代码错误所在大致位置有预先了解或估计。

输出中间变量法：在程序的需要观察中间变量结果的位置增加输出语句。通过对输出的中间结果来判断错误或大致定位错误所在的位置。此方相对简单，适合对错误比较明显或对错误所在位置大致进行定位，然后再用“debug+断点法”。

对程序的编码风格，请初学者注意：

- 1) 代码的缩进格式表达程序逻辑的层次，便于阅读和理解。if...else, for, while, 函数体、结构体定义等语句块采用缩进格式。讲究“代码格式与程序逻辑的统一”。
- 2) 同一语句块内，逻辑上自成段落的块之间应该用空行隔开。
- 3) 变量的命名尽量“见名知义”，尽量采用英文单词、英文缩写、中文拼音或拼音缩写。
- 4) 在代码中尽可能给予详细的注释。

10.11 如何阅读代码

从 main() 函数开始读起，根据函数的调用关系，理清模块之间的关系，弄清楚每个模块的功能，各个模块之间是如何配合完成指定功能的；明确明个变量的含义，注意变量的初值，以及变量值在什么情况下会发生变化，为什么能这样变；必须理解代码背后的算法的本质本质和数学模型；应该理解程序的适用范围：数据处理的范围，对输入数据的格式要求，假设和前提条件是什么。

10.12 注释与续行符

行注释符//将此符号后至行尾部分作为注释。

块注释符/* 被注释代码 */，能注释任意一块连续的代码。块注释符不能嵌套，能写在任何可以可出空格的地方。

注释不仅可用在为代码添加备注，也可用于代码调试。可用注释使某行或某块代码不起作用，以帮助调试。

作为续行符的\（反斜杠字符），它必须是被断行的最后一个字符，否则，如果斜杠后还有其它字符的话，斜杠被认为是转义字符了。续行符使得语句可以在任意的位置换行。

字符串常量中不能有续行符。尽量利用语句的自然间隔处换行。

在本部分的“宏定义#define”的例子展示了续行符的用法。

10.13 C 程序设计常见英文词表

序号	英文词	说明	序号	英文词	说明	
1	auto, automatic	自动的	33	compare, cmp	比较	关系运算
2	break	终止当前循环	34	equal	等于, 相等	
3	case	当...	35	greater than	大于	
4	char, character	字符类型	36	less than	小于	
5	const, constant	常量	37	unequal	不等于	
6	continue	继续	38	add	加	算术运算
7	default	缺省	39	arithmetic operation	算术运算	
8	do	做, 执行	40	dec, decrease	自减	
9	double, double precision	双精度浮点类型	41	div, divide	除	
10	else	否则	42	inc, increase	自增	
11	enum, enumulate	枚举	43	mod, modular	取余, 模运算	
12	extern	外部的	44	mul, multiple	乘	
13	float, float point	单精度浮点类型	45	operator, op	操作符	逻辑运算
14	for	循环结构	46	subtract, sub	相减	
15	goto	跳转至...	47	and	与	
16	if	如果	48	logic operation	逻辑运算	
17	inline	内联	49	not	非	
18	int, integer	整数类型	50	or	或	位运算
19	long	32bits 长整数类型	51	xor	异或	
20	register	寄存器	52	bitwise and	按位与	
21	static	静态的	53	bitwise not	按位非	
22	struct, structure	结构体	54	bitwise or	按位或	
23	switch	开关	55	bitwise xor	按位异或	格式控制
24	typedef, type define	类型定义	56	%c, character	字符	
25	union	联合体, 共用体	57	%d, decimal	十进制	
26	unsigned	无符号的	58	%f, float point	浮点数	
27	void	空、无, 待定	59	%lf, long float	双精度浮点数	
28	volatile	易失的	60	%lld, long long decimal	64 位整数	
29	while	循环结构, 当...	61	%o, octal	八进制数	
30	stderr, standard error file pointer	标准错误	62	%s, string	字符串	
31	stdin, standard input file pointer	标准输入	63	%x, hexadecimal	十六进制数	
32	stdout, standard output file pointer	标准输出	64	formart	格式, 格式化	

序号	英文词	说明	序号	英文词	说明
65	abort	终止	104	call	调用
66	append	(向末尾)追加	105	class	类, 班级
67	attach	附加	106	code	代码, 编码、码
68	break point	断点	107	col, column	列
69	build	构建	108	computer	计算机
70	compile	编译	109	concatenation, cat	拼接
71	compiler	编译器	110	condition	条件
72	debug	调试	111	copy, cpy	复制, 拷贝
73	debugger	调试器	112	count, cnt	计数
74	go	开始调试	113	data type	数据类型
75	link	连接	114	declare	声明
76	linker	连接器	115	define	定义
77	pause	暂停	116	delcared	已声明的
78	rebuild	重建	117	dimension, dim	维, 维度
79	run	运行	118	directory	目录
80	run to cursor	运行到光标处。	119	encode	编码
81	runtime error	运行时错误	120	end	结束
82	start	启动、开始	121	EOF, end of file	文件尾
83	step	单步	122	error	错误
84	stop	停止, 结束	123	exception	异常
85	access denied	拒绝访问	124	exit	退出
86	address	地址、寻址	125	file	文件
87	app, application	应用, 应用程序	126	file not found	找不到文件
88	argc, argumnet count	参数个数, main 的函数参数	127	find	查找
89	argv, argument variable	参数变量, main 的函数参数	128	flag	标志
90	array	数组	129	folder	文件夹
91	assign	赋值	130	function	函数
92	assignment	赋值	131	global	全局的
93	batch, bat	批量, 批处理	132	hardware	硬件
94	begin	开始	133	heap	堆
95	bianary tree	二叉树	134	identifier	标识符
96	binary	二进制	135	index	数组小标、索引
97	binary search	二分查找	136	index, idx	下标、索引
98	bit	二进制位	137	init, initiate	初始化
99	blank	空白	138	input	输入
100	bool, boolean	布尔类型	139	invalid	无效的
101	branch statement	分支语句	140	keyword	关键字
102	byte	字节	141	left shift	左移位
103	C language	C 语言	142	library, lib	库

序号	英文词	说明	序号	英文词	说明
143	linked list	链表	178	recursion	递归
144	list	链表	179	ref, reference	引用
145	local	局部的	180	return	返回
146	long long	64bits 长整数类型	181	right shift	右移位
147	loop	循环	182	row	行
148	main	主函数	183	search	查找, 搜索
149	math.h, math header	数学函数头文件	184	short, short integer	短整数类型
150	maximum,max	最大值	185	sign	符号
151	member	成员	186	signed	有符号的
152	memory	内存	187	sort	排序
153	middle,mid	中间的	188	sort	排序
154	minimum,min	最小值	189	software	软件
155	module	模块	190	source	源
156	multidimesion	多维	191	space	空格字符
157	nesting	嵌套	192	stack	栈
158	next	下一个, 接下来	193	statement	语句, 声明, 表述
159	node	结点	194	status	状态
160	null, NULL	空值, 空指针	195		
161	operand	操作数	196	storage	存储器
162	OS, operating System	操作系统	197	store	储存
163	output	输出	198	string, character string	字符串
164	overflow	溢出	199	string.h, string header	头文件, 字符串函数头文件
165	parameter	参数	200	subscript	下标
166	path	路径	201	sum	和, 累加和
167	pointer	指针	202	swap	交换、互换
168	position,pos	指针	203	system	系统
169	previous,pre	前一个	204	table,tab	表格, 跳格、跳格字符
170	priority	优先, 优先级	205	temp , tmp, temporary	临时的
171	program	程序	206	two-dimensional array	二维数组
172	programmer	程序员	207	type	类型
173	project	项目	208	undeclared	未声明的
174	query	查询	209	undefined	未定义的
175	random,rand	随机	210	value,val	值
176	read	读	211	variable,var	变量
177	real	实数, 真实, 现实	212	write	写