第一部分　语 言 参 考

第一章　词　　法

本章介绍 Borland C++词法的规范定义。C++词法和各种与单词类似的单元（即单词符号，token）有关，后者由语言本身所识别。相对而言，语法（见第二章）则详细地定义了组合单词符号以形成表达式、语句及其它单元的合法方式。

Borland C++中的单词符号是编译器及预处理器从用户程序的各种操作中抽取出来的。

使用恰当的文本编辑器（如 Borland C++编辑器），并通过击键输入源代码中的 ASCII 字符，就可以创建 Borland C++程序。Borland C++中的基本程序单元是文件，它通常与 RAM 或磁盘上的 DOS 文件相对应，其文件扩展名为.C 或者.CPP。

预处理器先扫描程序文本，以处理特殊的预处理器伪指令。例如，伪指令 #include <inc_file>可以在编译之前先将文件 inc_file 的内容添加（或包含）到程序中。同时，预处理器也扩展程序及包含文件中的任何宏定义。

1.1 空 白 符

在编译器的词法分析阶段，源代码文件被分解成单词符号和空白符。空白符指空格、水平制表符、垂直制表符、换行符及注释的集合。空白符可以指明单词符号从何处开始，到何处结束；但除此功能之外，其它任何空白符均被忽略。例如，下面两个序列

```
int i;  float f;
```

和

```
int i;
float f;
```

从词法上来讲是等价的，而且经过词法分析之后，得到的六个单词符号也相同：

(1) int
(2) i
(3) ;
(4) float
(5) f
(6) ;

表示空白符的 ASCII 字符可以出现在文字串中。在这种情况下，编译器不对其进行常规的词法分析处理；换言之，它们仍然作为串的一部分：

```
char name[ ] = "Borland International";
```

进行词法分析之后，得到七个单词符号，其中包括一个文字串单词符号：

```
"Borland International"
```

1.1.1 用\续行

如果碰到一行中最后的换行符前有一个反斜杠（\），那么这种情况要作特殊处理。反斜杠和换行符均被忽略，两个物理文本行被作为一行对待：

```
"Borland \
International"
```

被词法分析之后，得到"Borland International"。

1.1.2 注释

注释指用于解释程序的文本。注释只为程序设计者所使用，在进行词法分析之前，它们被从源程序文本中过滤掉。

有两种注释方式，即C方式和C++方式。Borland C++支持这两种形式的注释，而且还支持嵌套注释。在C程序或C++程序中，用户可以混合使用或单独使用任何一种注释。

1. C注释

C注释指符号偶/*之后的任意字符序列，直到碰到/*之后的第一个符号偶*/时注释才结束。在宏扩展之后，包含四个注释分界符在内的整个注释序列由一个空格所代替。注意，在某些C语言版本中，注释被直接删去，而不用空格代替。

Borland C++并不支持使用/***/方式的不可移植的单词符号分析策略。Borland C++采用ANSI规定的符号偶##，即：

```
#define VAR(i, j)  (i/**/j)          /* Won't work */
#define VAR(i, j)  (i##j)            /* OK in Borland C++ */
#define VAR(i, j)  (i ## j)          /* Also OK */
```

在Borland C++中，

```
int  /* declaration */  i  /* counter */;
```

被分析成

```
int  i;
```

给出的单词符号有三个，即：

```
int  i  ;
```

2. C++注释

用户也可在C源程序中用//产生注释行，这种注释方法仅用于Borland C++中。C++允许使用以两个相邻斜杠（//）作为分界符的单行注释。注释可以从任何位置开始，而且可以一直扩展到行尾为止：

```
class  X  {      // this is a comment
...};
```

3. 嵌套注释

ANSI C不支持嵌套注释。如果试图进行如下注释:

```
/* int /* declaration */ i /* counter */;  */
```

则会出错，因为注释的范围是第一个/*到其后的第一个*/之间的部分。这时，上一行即相当于

```
i;  */
```

当然会产生语法错误。

缺省情况下，Borland C++不支持嵌套注释，但用户可以使用编译选项达到这一目的。有关启动嵌套注释功能的信息，请参阅《用户指南》第三章。

4. 分界符及空白符

在少数情况下，在/*和//之前或在*/之后的空白符尽管从语法上来讲不是必需的，但它可以避免移植性方面的问题。例如，按照C语言约定，C++代码

```
int i = j // * divide by k */k;
+m;
```

经词法分析后即为 int i = j+m; 而不是我们所期望的

```
int i = j/k;
+m;
```

如果写成如下形式:

```
int i = j/  /* divide by k */  k;
+m;
```

则可以避免这类问题。

1.2 单词符号

Borland C++识别的单词符号有六类。单词符号的规范定义如下:

token:
- keyword (关键字)
- identifier (标识符)
- constant (常量)
- string-literal (串文字)
- operator (操作符)
- punctuator (分隔符，亦称为 separator)

在对源代码进行语法分析时，抽取单词符号的原则是，选取字符序列中最长可能的单词符号。例如，external 被作为单一的标识符对待，而不被分解成关键字 extern，后跟标识符 al。

有关单词符号分析的内容，在后面还要介绍。

1.2.1 关键字

关键字是为特定目的而保留的单词，不可将关键字用作常规的标识符名。表 1.1 到表 1.5 列出了 Borland C++中的关键字。用户可以利用选项（或命令行编译选项）仅选择 ANSI 关键字、UNIX 关键字或其它关键字。《用户指南》第一章和第三章中有关于选项的更详细的信息。

表 1.1 全部 Borland C++关键字

___asm	____es	interrupt	short
__asm	__es	____interrupt	signed
asm	____except	__interrupt	sizeof
auto	____export	____loadds	____ss
break	__export	__loadds	__ss
case	extern	long	static
catch	far	near	____stdcall
___cdecl	____far	__near	__stdcall
__cdel	__far	____near	struct
cdecl	____fastcall	new	switch
char	__fastcall	operator	template
class	____finally	____pascal	this
const	float	__pascal	____thread
continue	for	pascal	throw
____cs	friend	private	____try
__cs	goto	protected	try
default	huge	public	typedef
delete	____huge	register	union
do	huge	__huge	return
unsigned	double	if	__rtti
virtual	____ds	____import	____saveregs
viod	__ds	__import	__saveregs
volatile	else	inline	____seg
while	enum	int	__seg

表 1.2 Borland C++中的寄存器变量

__AH	__CL	__EAX	__ESP
__AL	__CS	__EBP	__FLAGS
__AX	__CX	__EBX	__FS
__BH	__DH	__ECX	__GS
__BL	__DI	__EDI	__SI
__BP	__DL	__EDX	__SP
__BX	__DS	__ES	__SS
__CH	__DX	__ESI	

表 1.3 Borland C++对 ANSI C 的扩充

____asm	____except	____import[1]	pascal
__asm	__export	__import[1]	____saveregs[1]
____cdel	__export	____interrupt[1]	__saveregs[1]
__cdecl	____far[1]	__interrupt[1]	____seg[1]
cdecl	__far[1]	interrupt[1]	____seg[1]
____cs[1]	far[1]	____loads[1]	__seg[1]
__cs[1]	____fastcall	__loadds[1]	__ss[1]
____ds[1]	__fastcall	____near[1]	__ss[1]
__ds[1]	____finally	__near[1]	____rtti
____es[1]	____huge[1]	near[1]	____thread[2]
__es[1]	__huge[1]	____pascal	____try
huge[1]	__pascal		

[1] 仅用于 16 位编译器。

[2] 仅用于 32 位编译器。

表 1.4 C 中特有的关键字

____finally	____try

表 1.5 C++中特有的关键字

asm	friend	protected	try
catch	inline	public	virtual
class	new	template	____rtti
delete	operator	this	private
throw			

1.2.2 标识符

标识符的规范定义如下：

identifier:
    nondigit
    identifier nondigit
    identifier digit
nondigit: one of
    a b c d e f g h i j k l m n o p q r s t u v w x y z __
    A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
digit: one of
    0 1 2 3 4 5 6 7 8 9

1. 命名及长度限制

标识符指具有任意长度的任意类名、对象名、函数名、用户自定义的数据类型名及其它名称。在标识符中可以出现字母 A 到 Z 及 a 到 z、下划线（_）和数字 0 到 9。其限制只有两种:

(1) 首字符必须是字母或者下划线。

(2) 缺省情况下，Borland C++只识别标识符中前32个字符。可以利用菜单或命令行选项减少有效字符数，但不能增加它。有关这些选项的信息，可参阅《用户指南》第一章和第三章。

2. 标识符及其大小写

在 Borland C++中，大小写是区别对待的，所以，Sum，sum 和 suM 是三个不同的标识符。

在其它模块中定义的全程变量遵从同样的常规标识符的命名和长度规则。但是，为了在链接时与不区分大小写的语言兼容，Borland C++中提供了不区别对待大小写字母的选项。使用这些选项时，全程变量 Sum 和 sum 将被看成同一标识符，这样，在链接过程中可能会产生警告信息"Duplicate symbol"。

有关链接和区别对待大小写标识符的选项，请参阅《用户指南》第一章和第三章。

例外的情况是，在链接时，__pascal 类型的标识符通常全部被转换成大写形式。

3. 唯一性和作用域

尽管标识符的名字可以是任意的（正如规则本身所示），但如果在同一个作用域和同一个名空间中，多个标识符采用相同的名字，则会出现错误。不管作用域如何，在不同的名空间中，重复名通常都是合法的。后面还要讨论作用域规则。

1.2.3 常量

常量是用以表达有固定数值或字符值的单词符号。Borland C++支持四种类型的常量：浮点数、整数、枚举型常量及字符常量(包括字符串)。后面的图 1.1 说明了这些常量的内部表示。

编译器根据常量的数字值及其在源代码中的格式来确定常量的数据类型。常量的规范定义见表 1.6。

1. 整型常量

整型常量可以是十进制（以十为基数）、八进制（以八为基数）或十六进制（以十六为基数）整数。如果整数中没有任何后缀，则如表 1.7 所示，根据整数的值确定整型常量的数据类型。注意，对于十进制和非十进制常量而言，该规则是有区别的。

表 1.6　常量的规范定义

```
constant:
    floating–constant
    integer–consant
    enumeration–constant
    character–constant

floating–constant:
    fractional–constant <exponent–part> <floating–suffix>
    digit–sequence exponent–part <floating–suffix>

fractional–constant:
    <digit–sequence>.digit–sequence
    digit–sequence.

exponent–part:
    e <sign> digit–sequence
    E <sign> digit–sequence

sign: one of
    + –

digit–sequence:
    digit
    digit–sequence digit

floating–suffix: one of
    f l F L

integer–constant:
    decimal–constant <integer–suffix>
    octal–constant <integer–suffix>
    hexadecimal–constant <integer–suffix>

decimal–constant:
    nonzero–digit
    decimal–constant digit

octal–constant:
    0
    octal–constant octal–digit
```

(续表)

hexadecimal-constant:
0 x hexademical-digit
0 X hexadecimal-digit
hexadecimal-constant hexadecimal-digit

nonzero-digit: one of
1 2 3 4 5 6 7 8 9

octal-digit: one of
0 1 2 3 4 5 6 7
a b c d e f
A B C D E F

integer-suffix:
unsigned-suffix <long-suffix>
long-suffix <unsigned-suffix>

unsigned-suffix: one of
u U

long-suffix: one of
I L

enumeration-constant:
identifier

character-constant:
c-char-sequence

c-char-sequence:
c-char
c-char-sequence c-char

c-char:
Any character in the source character set except the single-quote (') , backslash (\) ,or newline characcter escape-sequence.

escape-sequence: one of
\" \' \ \\
\a \b \f \n
\o \oo \ooo \r
\t \v \Xh··· \xh···

(1) 十进制常量

十进制常量值的范围是从 0 到 4 294 967 295。超过此范围限制的整型常量将被截断。十进制常量的前面不能出现数字"0"；有前导"0"的整型常量被解释成八进制常量。因而，

```
int  i = 100;          /* decimal 100 */
int  i = 010;          /* decimal 8 */
int  i = 0;            /* decimal 0 = octal 0 */
```

(2) 八进制常量

有前导零的所有常量均被作为八进制常量对待。如果八进制常量中发现了不合法的数字 8 或 9，则报告错误信息。超过 037777777777 的八进制常量将被截断。

(3) 十六进制常量

以 0x（或 0X）开头的所有常量均被作为十六进制常量对待。超过 0xFFFFFFFF 的十六进制常量将被截断。

(4) 长整数及无符号数后缀

任何其后附以后缀 L（或 l）的常整数都被视为长整数（long）。同样，在常整数后面附以后缀 U（或 u）即可使之成为无符号整数（unsigned）。不管常量为多少进制，只要其本身的值大于 65 535，则都被称为 unsigned long 型整数。可以在同一常量后面同时使用 L 后缀和 U 后缀，且大小写及其次序也可以是任意的，如 ul，lu，UL 等。

表 1.7　不带 L 或 U 的 Borland C++整型常量

十进制常量	
0 to 32767	int
32768 to 2147483647	long
2147483648 to 4294967295	unsigned long
> 4294967295	将被截断
八进制常量	
00 to 077777	int
01000000 to 0177777	unsigned int
02000000 to 017777777777	long
020000000000 to 037777777777	unsigned long
> 037777777777	将被截断
十六进制常量	
0x0000 to 0x7FFF	int
0x80000 to 0xFFFF	unsigned int
0x100000 to 0x7FFFFFFF	long
0x800000000 to 0xFFFFFFFF	unsigned long
> 0xFFFFFFFF	将被截断

在没有后缀 U，u，L 或 l 时，常量数据的数据类型是可以调节其值的下列类型之

Decimal	int，long int，unsigned long int
Octal	int，unsigned int，long int，unsigned long int
Hexadecimal	int，unsigned int，long int，unsigned long int

如果常量的后缀为 U 或 u，则其数据类型为 unsigned int，unsigned long int 中可以容纳其值的第一个类型。

如果常量的后缀为 L 或 l，则其数据类型为 long int，unsigned long int 中可以容纳其值的第一个类型。

如果常量的后缀为 u 和 l (ul，lu，Ul，lU，uL，Lu，LU 或 UL)，则其数据类型为 unsigned long int。

表 1.7 中汇总了三种进制（十进制、十六进制和八进制）下的整型常量的表示方法；假定所指定的数据类型中没有使用后缀 L 或 U。

2. 浮点常量

浮点常量由五部分组成：

(1) 十进制整数部分
(2) 小数点
(3) 十进制小数部分
(4) e 或 E 及一个带符号的整型指数（可选）
(5) 类型后缀：f，F，l 或 L（可选）

可以省略十进制整数或十进制小数部分，也可以将这两部分全部省略掉。可以省略小数点或字符 e（或 E）和带符号的整型指数（但不可同时省略两者）。这些规则适用于浮点数的常规表示法和科学（指数）计数法。

负的浮点常量被认为是正的浮点常量前置一个负号 (–)。例如：

常量	值
23.45e6	23.45×10^{6}
.0	0
0.	0
1.	$1.0\times10^{0}=1.0$
–1.23	–1.23
2e–5	2.0×10^{-5}
3E+10	3.0×10^{10}
.09E34	0.09×10^{34}

如果浮点常量的后面带有任何后缀，则此浮点常量的类型为 double。但是，如果在浮点常量的后面置以后缀 f 或 F，则可以将此浮点常量强制为 float 类型。同样地，用后缀 l 或 L 可以将浮点常量强制为 long double 类型。表 1.8 列出了 float，double 和

long double 类型浮点数的范围。

表 1.8 Borland C++浮点常量的大小和范围

类型	大小（位数）	范围
float	32	3.4×10^{-38} 到 3.4×10^{38}
double	64	1.7×10^{-308} 到 1.7×10^{308}
long double	80	3.4×10^{-4932} 到 1.1×10^{4932}

3. 字符常量

字符常量指处于单引号中的一或多个字符，如'A'，'='或'\n'等。在 C 语言中，单字符常量的类型为 int，其内部表示占 16 位（两个字节），高位字节为 0 或带扩展的符号位。在 C++中，字符常量的类型为 char。在 C 和 C++中，多字符常量的数据类型均为 int。

为了比较字符类型的大小，可分别将下列代码作为 C 和 C++程序来编译：

```
#include <stdio.h>
#define CH 'x'              /* A CHARACTER CONSTANT */
void main(void) {
   char ch = 'x';           /* A char VARIABLE */

   printf("\nSizeof int       = %d", sizeof(int));
   printf("\nSizeof char      = %d", sizeof(char));
   printf("\nSizeof ch        = %d", sizeof(ch));
   printf("\nSizeof CH        = %d", sizeof(CH));
   printf("\nSizeof wchar_t   = %d", sizeof(wchar_t));
}
```

程序的运行结果见表 1.9(单位为字节)。

表 1.9 字符类型的大小

作为 C 程序的输出			作为 C++程序的输出		
	16 位	32 位		16 位	32 位
Sizeof int	= 2	4	Sizeof int	= 2	4
Sizeof char	= 1	1	Sizeof char	= 1	1
Sizeof ch	= 1	1	Sizeof ch	= 1	1
Sizeof CH	= 2	4	Sizeof CH	= 1	1
Sizeof wchar_t	= 2	2	Sizeof wchar_t	= 2	2

(1) 三字符类型

像'A'，'\t'和'\007'等单字符常量表示为 int 类型的值。这种情况下，低位字节被符号扩展到高位，即当值大于 127（十进制）时，高位被置为-1（=0xFF）。若将缺省类型 char 声明成 unsigned，那么，不管低位的值为多少，高位均被置为 0。有关这些选项的信息，请参阅《用户指南》中第一章和第三章。

三种字符类型 char，signed char 和 unsigned char 都占八位（一个字节）。对于用 Borland C++ 4.0 以前的版本开发的 C 和 Borland C++程序，char 与 unsigned char 是一样的。C 程序的功能不受这三种不同字符类型的影响。

注：为了保留旧的功能，可使用-K2 命令行选项和 Borland C++ 3.1 头文件。

在 C++程序中，可以用 char，signed char 和 unsigned char 类型的参数对函数进行重载。例如，下列函数原型是有效的，而且互不相同：

```
void func(char ch);
void func(signed char ch);
void func(unsigned char ch);
```

若上述任何一个原型存在，就可以接受三种字符类型。例如，下列程序是合法的：

```
void func(unsigned char ch);
void main(void) {
    signed char ch = 'x';
    func(ch);
}
```

有关代码生成选项的信息，请参阅《用户指南》中第一章和第三章。

(2) 转义序列(特定于 C++)

反斜杠字符（\）用于引入转义字符序列，在其后面常跟以可见的非图形字符。例如，常量\n 用于表示单一的换行符。

反斜杠可以和八进制或十六进制数值结合起来使用，以表示相应于该数值的 ASCII 符号或控制码。例如，'\03'代表 Ctrl-C，'\x3F'表示问号。在转义序列中可以使用多达三个八进制数字串或任意数目的十六进制数字，只要该数值的大小不超过数据类型 char 的限制（对于 Borland C++而言，即 0 到 0xff 之内）。数值太大时会产生编译错误 "Numeric constant too large"。例如，八进制数值\777 比所允许的最大值\377 大，因而会导致编译错误。在八进制或十六进制转义序列中所碰到的第一个非八进制或非十六进制字符标志着该转义序列的结束。

实际上，Turbo C 仅允许在十六进制转义序列中使用三位数字。在 Borland C++中采用的 ANSI C 规则可能会导致由旧版本代码带来的问题，因为旧版本代码假定只转换前三个字符。例如，要使用 Turbo C 1.x 定义一个响铃串（ASCII 7），且后跟一序列数字字符，程序员可能会编写如下语句：

```
printf ("\x0072.1A Simple Operating System");
```

上述语句的期望解释为\x007 及"2.1A Simple Operating System"，但是 Borland C++编译器将它解释为十六进制数值\x0072 和文字串".1A Simple Operating System"。

为了避免这类问题，可以将上述代码改写如下：

```
printf ("\x007" "2.1A Simple Operating System");
```

当八进制转义序列后跟以非八进制数字值时，也会出现二义性问题。例如，由于 8 和 9 不是合法的八进制数字，这样常量\258 将被解释成由两个字符\25 和 8 组成的字符常

量。

表 1.10 列出了可用的转义序列。

表 1.10 Borland C++转义序列

序列	值	字符	功能
\a	0x07	BEL	响铃警告
\b	0x08	BS	退格
\f	0x0C	FF	走纸
\n	0x0A	LF	换行
\r	0x0D	CR	回车
\t	0x09	HT	水平制表
\v	0x0B	VT	垂直制表
\\	0x5C	\	反斜杠
\'	0x27	'	单引号
\"	0x22	"	双引号
\?	0x3F	?	问号
\O		any	O=三个八进制数字组成的数串
\xH		any	H=十六进制数字组成的数串
\XH		any	H=十六进制数字组成的数串

注: \\可表示实际的 ASCII 反斜杠，并可用于系统路径中。

(3) 宽字符常量

宽字符常量可以表示不适用于 char 类型存储空间的字符。宽字符常量占两个字节。前置 L 的字符常量为宽字符常量，其数据类型为 wchar__t（在 stddef.h 中定义的一种整数类型)。例如:

```
wchar__t  x = L'AB';
```

前置 L 的字符串称为宽字符串，其中每个字符占两个字节的空间。例如:

```
wchar__t  str = L"ABCD";
```

(4) 多字符常量

Borland C++也支持多字符常量。当使用 32 位编译器时，多字符常量可由多达四个字符组成。16 位编译器只支持双字符常量。例如，'An'，'\n\t'和'\007\007'等在 16 位程序中均是合法的，而常量'\006\007\008\009'只在 32 位程序中是合法的。使用 16 位编译器时，这些常量被表示成 16 位 int 值，其中第一个字符在低位字节上，第二个字符在高位字节上。对于 32 位编译器，这些常量被表示成 32 位 int 值。这些常量不可移植到其它 C 编译器中。

4. 串常量

串常量亦称为串文字，它可以形成用于处理固定字符序列的常量类型。串常量的数据类型为 array-of-char(字符数组)，其存储类为 static，可以写成处于双引号之中的任意数目的字符序列:

```
"This is literally a string!"
```

空串可以写成""。

双引号内的字符可以含有字符的转义序列。例如，下列代码

```
"\t\t\"name\"\\\tAddress\n\n"
```

打印出来就是:

```
"Name" \          Address
```

"Name" 前有两个制表符，Address 前有一个制表符。该行后面有两个空白行。\" 可以表示内部的双引号。

如果为了与 ANSI 兼容而用-A 选项编译，那么编译器就将字符序列"\\"转换成"\"。

文字串被内部存储成给定的字符序列，后加一个空字符 ('\0')。空串则被存储成单一的'\0'字符。

在语法分析过程中，仅以空白符隔开的相邻文字串被并在一起，以形成一个新的文字串。在下列例子中，

```
#include <stdio.h>
#include <windows.h>

#progma argsused

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance, LPSTR
                    lpszCmdParam, int nCmdShow)
{
    char  * p;
    P = "This is an example of how Borland C++"
        "will automatically\ndo the concatenation for"
        "you on very long strings, \nresulting in nicer"
        "looking programs.";
    printf (p);
    return (0);
}
```

程序的输出形式为:

```
This is an example of how Borland C++ will automatically
do the concatenation for you on very long strings,
resulting in nicer looking program.
```

也可以使用反斜杠 (\) 作为续行符，将某个串常量与下一行的串常量合并在一起:

```
puts  ("This is really \
a one-line string");
```

5. 枚举型常量

枚举型常量是用类型说明符 enum 定义的标识型。为了提高清晰性和可读性，枚举型常量常用便于记忆的标识符表示。枚举型常量的数据类型是整型。在可合法使用整型常量的任意位置均可以使用枚举型常量。在 enum 说明的作用域内，所用的标识符必须是唯一的。允许出现负的初始值。

枚举型常量所要求的数值依赖于枚举说明的格式及可选的初始化语句。在例子

```
enum team {giants, cubs, dodgers};
```

中，giants，cubs 和 dodgers 是 team 型的枚举常量，可以将其赋给 team 类型的任何变量或整数类型的任何其它变量。枚举型常量所要求的值是

```
giants = 0, cubs = 1, dodgers = 2
```

这里没有显式的初始化语句。在下列例子中，

```
enum team {giants, cubs = 3, dodgers = giants+1};
```

枚举型常量被设置如下：

```
giants = 0, cubs = 3, dodgers = 1
```

常量值不必是唯一的：

```
enum team {giants, cub = 1, dodgers = cubs-1};
```

表 1.11　16 位数据类型、大小和范围

类型	大小(位数)	范围	应用实例
unsigned char	8	0 到 255	较小的数及 PC 字符全集
char	8	−128 到 127	非常小的数及 ASCII 字符
enum	16	−32 768 到 32 767	数值的有序集合
unsigned int	16	0 到 65 535	较大的数和循环
short int	16	−32 768 到 32 767	计数、较小的数、循环控制
int	16	−32 768 到 32 767	计数、较小的数、循环控制
unsigned long	32	0 到 4 294 967 295	天文距离
long	32	−2 147 483 648 到 2 147 483 647	较大的数、人口数目
float	32	-3.4×10^{-38} 到 3.4×10^{38}	科学计数（精确到 7 位）
double	64	1.7×10^{-308} 到 1.7×10^{308}	科学计数（精确到 15 位）
long double	80	3.4×10^{-4932} 到 1.1×10^{4932}	财政开支（精确到 19 位）
near pointer	16	Not applicable	处理内存地址
far pointer	32	Not applicable	处理当前段外的内存地址