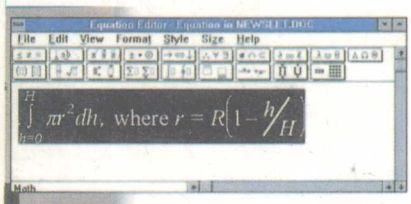
Windows技术丛书之四

MS-DOS 和 Windows 环境下的 C++语言编程指导

李朝阳　夏　伟　郭　永　郭建中
赵　丽　刘　军　任　军　李梅君　编

海洋出版社

4



Windows 软件技术丛书之四

MS-DOS 和 Windows 环境下的 C++语言编程指导

李朝阳　夏　伟　郭　永　郭建中
赵　丽　刘　军　任　军　李梅君　　编
徐　倩　林　红　吴　刚　　审校

海 洋 出 版 社
1993 年

内 容 提 要

本书共分十三章三个附录，对C++语言进行了全面而详细的说明．其内容包括：C++基本元素，表达式，语句，说明，类，重载，友元，多态性，预处理等，这些内容是C++语言的基础部分，也是掌握C++语言的关键．

本书是了解和掌握C++语言的最佳选择用书，适用于从事计算机软件及应用开发的技术人员，亦可供在校师生及自学者使用．

欲购本书的用户请直接与北京8721信箱联系，电话：2562329，邮政编码：100080．

（京）新登字087号

责任编辑：闫世尊

Windows软件技术丛书之四

MS-DOS和Windows环境下的C++语言编程指导

李朝阳　夏　伟　郭　永　郭建中
赵　丽　刘　军　任　军　李梅君　　编
徐　倩　林　红　吴　刚　　审校

海洋出版社出版（北京市复兴门外大街1号）
海洋出版社发行　　兰空印刷厂印刷
开本：787×1092　1／16　印张：13.6　字数：350千字
1993年12月第一版　　1993年12月第一次印刷
印数：1—5000册

*

ISBN 7-5027-3821-5／TP・234　　定价：193.00元／套(6册)

目　录

简介 …… 1
第一章　词法约定 …… 3
1.1　标记 …… 3
1.2　注释 …… 4
1.3　标识符 …… 5
1.4　C++关键字 …… 6
1.5　标点符号 …… 7
1.6　算符 …… 7
1.7　文字 …… 11
第二章　基本概念 …… 18
2.1　术语 …… 18
2.2　说明与定义 …… 18
2.3　作用域 …… 19
2.4　程序与链接 …… 23
2.5　启动与终止 …… 26
2.6　存储类 …… 31
2.7　类型 …… 34
2.8　左值与右值 …… 42
2.9　名空间 …… 42
2.10　最大值与最小值 …… 42
第三章　标准转换 …… 45
3.1　整数拓宽 …… 45
3.2　整数转换 …… 45
3.3　浮点转换 …… 47
3.4　浮点与整型转换 …… 47
3.5　算术转换 …… 47
3.6　指针转换 …… 48
3.7　引用转换 …… 51
3.8　成员指针转换 …… 51
第四章　表达式 …… 52
4.1　表达式的类型 …… 52
4.2　表达式的语义 …… 81
4.3　表达式中的注释 …… 83
第五章　语句 …… 84
5.1　概述 …… 84

5.2 标号语句 …… 84
5.3 表达式语句 …… 86
5.4 复合语句(块) …… 87
5.5 选择语句 …… 87
5.6 递归语句 …… 90
5.7 转移语句 …… 93
5.8 说明语句 …… 95
第六章 说明 …… 99
6.1 指明符 …… 99
6.2 枚举说明 …… 110
6.3 链接规约 …… 114
第七章 说明符 …… 117
7.1 概述 …… 117
7.2 类型名 …… 119
7.3 抽象说明符 …… 120
7.4 函数定义 …… 138
7.5 初始值 …… 140
第八章 类 …… 147
8.1 概述 …… 147
8.2 类名 …… 150
8.3 类成员 …… 152
8.4 成员函数 …… 155
8.5 静态数据成员 …… 161
8.6 联合 …… 161
8.7 位域 …… 164
8.8 嵌套的类说明 …… 165
8.9 类作用域中的类型名 …… 168
第九章 派生类 …… 169
9.1 概述 …… 169
9.2 多重基类 …… 175
9.3 虚函数 …… 180
9.4 抽象类 …… 183
9.5 作用域规则小结 …… 185
第十章 成员存取控制 …… 187
10.1 控制对类成员的存取 …… 187
10.2 存取指明符 …… 187
10.3 基类的存取指明符 …… 188
10.4 友元 …… 191
10.5 受保留的成员存取 …… 194

10.6 存取虚函数 …… 194
10.7 多重存取 …… 195
第十一章 特殊的成员函数 …… 197
11.1 构造函数 …… 198
11.2 析构函数 …… 201
11.3 临时对象 …… 204
11.4 转换 …… 205
11.5 new 与 delete 算符 …… 208
11.6 用特殊的成员函数初始化 …… 214
11.7 复制类成员 …… 218
第十二章 重载 …… 221
12.1 概述 …… 221
12.2 说明匹配 …… 222
12.3 变元匹配 …… 224
12.4 重载函数的地址 …… 229
12.5 重载算符 …… 229
第十三章 预处理 …… 238
13.1 预处理程序 …… 238
13.2 宏 …… 239
13.3 包含文件 …… 224
13.4 条件编译 …… 246
13.5 行控制 …… 250
13.6 报错指令 …… 250
13.7 #pragma 指令 …… 251
附录 A 翻译阶段 …… 256
附录 B 特定修饰符 …… 257
B.1 存储模式修饰符 …… 258
B.2 调用与命名约定修饰符 …… 269
B.3 特殊修饰符 …… 272
附录 C 语法小结 …… 274
C.1 关键字 …… 274
C.2 表达式 …… 274
C.3 说明 …… 277
C.4 说明符 …… 279
C.5 类 …… 280
C.6 语句 …… 282
C.7 预处理程序 …… 282
C.8 扩展 …… 282

简　介

本书主要介绍:

可移植的操作系统接口(文件和屏幕 I／O); 串和缓冲器处理; 浮点算术变换; 字符分类信息和其它支持功能。

对已掌握 C++编程基础的程序员尤为适用。

本书由十三章和三个附录组成。

第一章"词法约定"介绍 C++程序的基本元素。

第二章"基本概念", 讲解诸如作用域、链接、程序启动和终止、存储类和类型等概念。

第三章"标准转换", 描述编译程序在基本类型间所执行的类型转换。此外, 还讲解了编译程序如何在指针、引用和指向成员类型的指针之间进行转换。

第四章"表达式", 描述 C++表达式适用于计算值、标识对象或函数或生成副作用的算符序列和操作数。

第五章"语句", 讲解控制程序如何以及以何种顺序执行的 C++程序元素。所涉及的语句包括表达式语句、复合语句、选择语句、递归语句、转移语句、说明语句以及语句。

第六章"说明", 用于讲解完整说明如何用于构成说明语句。这一章涉及这样的主题: 存储类说明符、函数定义、初始化、枚举、类、结构和联合说明, 以及 typedef 说明。有关信息可参见第七章"说明符"和附录 B"特定的修饰符"。

第七章"说明符", 讲解命名对象、类型或函数的说明语句部分。

第八章"类", 介绍 C++类。使用 class, struct 或 union 关键字说明的对象在 C++中被视为类类型, 这一章讲解如何使用这些类类型。

第九章"派生类", 详述继承的概念。

第十章"成员存取控制", 详述如何控制对类成员的存取。使用存取控制说明符有助于产生更为完善的代码, 因为我们可以限制对象状态可能更改的多种方式。

第十一章"特殊的成员函数", 描述唯一于类类型的特殊函数。这些特殊函数执行初始化(构造函数)、清除(析构函数)和转换操作。这一章还描述 new 和 delete 算符, 这两个算符适用于动态存储器分配。

第十二章"重载", 解释在 C++中如何用同名但不同变元定义函数组。调用函数组中哪个函数依据实际函数调用中的变元列表, 此外, 这一章还涉及重载算符。

第十三章"预处理", 描述 C++预处理程序以及由 C++认可的编译指令。

附录 A"翻译阶段", 讲解 C++程序以何种顺序从源代码翻译成可执行文件。

附如 B"特定修饰符", 描述特定于 C++的修饰符。这些修饰符控制存储器寻址、调用约定等。

附录 C"语法小结", 含有扩展的 C++语法。

本书使用如下约定:

举例	描述
STDIO.H	大写字母表示在操作系统命令级上所用的文件名, 段

	名，寄存器和术语。
grammar–element$_{opt}$	opt下标表示语法的这一元素是可选的，可以省略。
[options]	方括号中的项是可选的。
pragmapack{1|2}	花括号和垂直线表示选择两个或多个项目之一。
CL [options…] file…	项后面的三点(省略号)表示可能出现多个拥有相同形式项。
while ( )	表示举例程序部分有意省略。
CTRL+ENTER	大写字母用于表示键盘上的键名。当见到两个键名间的加号(+)时，应在按下前一个键时，再按后一个键。回车键有时在键盘上表示为弯曲箭头，称为 ENTER。
″argument″	当第一次定义新项时，使用双引号。
″Cstring″	像串这样的语言结构需要引号。本语言所需的引号具有″″和′′形式。

在本书中的术语“变元”指传送给函数的实体。在有些情况中，用“实际”或“形式”修饰。

术语“变量”指简单的 C 类型数据对象。术语“对象”指 C++对象和变量。

有关其它术语的详细信息请见第二章中的“术语”一节。

第一章 词法约定

这一章介绍 C++程序的基本元素，它们对编译程序很有意义。这些元素称为词法元素，用于构造语句、定义、说明等等，后者构造完整程序。这些元素是：标记、注释、标识符、C++关键字、标点、算符、文字。

C++程序与 C 程序一样，由一个或多个文件构成，其中每个文件是按如下顺序翻译的：

1) 词法标记。在该翻译阶段，执行字符变换和三字母代码处理、接行和标记。

2) 预处理。预处理指令始终以#字符开头。一行只能有一条预处理指令。

例如：

```
#include <iostream.h>   // Include text of iostream.h in
                        //  translation unit.
#define NDEBUG          // Define NDEBUG (NDEBUG contains empty
                        //  text string).
```

3) 代码生成。该翻译阶段使用在预处理阶段中所生成的标记来生成对象代码。

在该阶段，执行源代码的语法检查和语义检查。

有关源程序如何翻译的详细信息见附录 A“翻译阶段”。

注释：C++预处理程序是 ANSI C 预处理程序的超集。在对单行注释、___cplusplus 常量的定义支持方面以及在对 C++算符的支持方面有所不同：

1) .*

2) ->*

3) ::

有关算符和注释的详细信息见“算符”和“注释”两节。

1.1 标 记

标记是 C++程序中对编译程序的最小元素。C++分析程序认可以下标记：标识符、关键字、文字、算符和其它分隔符。这些标记流构成了翻译单位。

标记常由空白间隔分隔。空白间隔可以是：空白、水平制表符或垂直制表符、换行、换页、注释。

标记包括：关键字、标识符、常量、算符、标点。

预处理标记包括：头名、标识符、PP-数字、字符常量、串文字、算符、标点、不是上述字符的非空白间隔字符。

分析程序通过用输入字符创建最长的标记将标记与输入流分隔开。考虑如下代码段：

```
a = i+++j;
```

程序员也许有如下一种想法：

1) j先增值，将i和j的值相加，然后将和赋给a(在此，标记为i，+和++j)。有关增值的信息见第四章中的“增值与减值算符”一节。

2) 将i和j的值相加，将和赋给a，之后i增值(在此，标记为i，++，+和j)。

这一解释等效于 a＝(i++)+j。

由于分析程序有可能根据输入流创建最长的标记，所以它选择第二种解释，使标记为i++和 j。

1.2 注 释

编译程序忽略注释但注释对程序员非常有用。注释通常用于为方便参阅注释代码。编译程序视它们为空白间隔。不过，有时为了检测注释，提供某些代码行，但最好使用#if／#end if 预处理命令。

C++注释有两种书写形式:

1) ／＊(斜线，星号)字符，后面是任意字符序列(包括换行)，然后是＊／字符。该语法与 ANSI C 相同。

2) ／／(两个斜线)字符，后面跟任意字符序列。不能用换行符终止这种形式的注释。因此，常称这种注释形式为单行注释。

注释字符(／＊ ＊／和／／)在字符常量、串文字或注释中没有特殊的含义。因此，使用第一种注释方式的注释不能嵌套。请考虑如下举例:

```
/* Intent:  Comment out this block of code.
   Problem: Nested comments on each line of code are illegal.
FileName = String( "hello.dat" ); /* Initialize file string */
cout << "File: " << FileName << "\n"; /* Print status message */
*/
```

编译程序将不会编译上面的代码，因为编译程序从第一个／＊到第一个＊／扫描输入流，并视其为注释。在该例中，第一个＊／出现在“Initialize file string”注释的结尾。然后，最后一个＊／不再与前头的＊／相匹配。

注意: 后面跟有续行符(＼)的单行注释形式具有特殊的功能。考虑如下代码:

```
#include <stdio.h>

int main()
{
    printf( "This is a number %d", // \
             5 );

    return 0;
}
```

在预处理后，上面的代码变为如下形式:

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

```
int main()
{
    printf( "This is a number %d", // 5 );

    return 0;
}
```

由于单行注释将一个逻辑行上的所有文本视为注释，所以上面的程序产生报错信息。

表 1.1 C / C++ 预定义标识符

标识符	兼容性	值
__cplusplus	C++	该宏的值无意义．如果定义它的话，程序作为 C++编译，对于作为 C 编译的翻译单元不定义该宏．
__DATE__	ANSIC，C++	源文件的编译日期．日期是"Mmm dd yy" 形式的字符串．包含引号是为了形成适当的 C++串．
__FILE__	ANSI C，C++	当前源文件名．__FILE__ 扩展为由双引号括起来的串．
__LINE__	ANSIC，C++	当前源文件的行号．行号是十进制数．
__STDC__	ANSI C	只有使用 / Za 编译选项才等于 1．
__TIME__	ANSI C，C++	源文件的编译时间．时间是"hh：mm：ss"形式的字符串．含有引号以形式适当的 C++串．
_MSC_VER		以 ddd 形式定义编译程序版本为串文字．
_MSDOS		始终定义．标识符将操作系统视为 MS_DOS．
__timestamp		当前源文件的翻译日期和时间．其字符串形式采用"Ddd Mmm dd hh：mm：ss" 形式．
_M_I86		始终定义．标识符视机器为 8086 系列．
_M_I8086		当编译程序把 8086 和 8088 处理器作为目标时，定义．
_M_I286		当编译程序把 80286 处理器作为目标时，定义．
_M_I386		当编译程序把 80386 处理器作为目标时，定义．
_M_I86mM		始终定义．标识存储模式，其中 m 可是 S（小模式），M（中模式），L（大模式），H（巨大模式）．如果使用巨大模式，则定义_M_I86LM 和_M_I86HM．小模式为缺省．有关存模式的详细信息，见附录 B"特定修饰符．"
___DLL		为运行时库定义为 DLL（/ MD 编译选项）．
NO_EXT_KEY		在早期版本中曾定义了该宏．现不再使用．
_CHAR_UNSIGNED		仅当给定

1.3 标识符

标识符是用于表示这样一些字符序列：对象或变量名；类、结构或联合标记；枚举类型；类成员；结构成员；联合成员或枚举成员；函数或类成员函数；typedef 名；标号名；宏名；宏参数。其语法为：

标识符：{非数字 / 标识符 非数字 / 标识符 数字}

非数字可以是：{_ *abcdefghijklmnopqrstuvwxyz* / *ABCDEFGHIJKLMNOPQRSTUVWXYZ*}

数字可以是：0 1 2 3 4 5 6 7 8 9

C++标识符是字符序列，必须少于247个字符(即只有前247个字符有意义)。影响标识符长度的因素是：标识符是否表示用户定义类型的对象；标识符是否表示函数；函数的变元个数；标识符是否命名用户定义类型对象。

标识符的第一个字符必须是大小字母或下划线(_)。由于C++标识符区分大小写，所以fileName有别于File Name。

标识符不能与关键字拼法相同，大小写相同(详见"C++关键字"一节)。含有关键字的标识符是合法的。例如，Print是合法标识符，尽管它含有int关键字。

在标识符开头连续使用两个下划线(__)或使用一个前导下划线后面跟大写字母都是为C++实现保留的。应防止使用一个前导下划线，后面跟小写字母，因为这有可能与当前或将来的保留标识符发生冲突。

1.4 C++关键字

关键字是预定义的保留标识符，具有特殊含义。在程序中，它们不能用作标识符。

下面关键字的第二部分特定于C++。当在编译时使用/Za选项时，禁止使用这些关键字。

asm	float	signed
auto	for	sizeof
break	friend	static
case	goto	struct
catch	if	switch
char	inline	template
class	int	this
const	long	throw
continue	new	try
default	operator	typedef
delete	private	union
do	protected	unsigned
double	public	virtual
else	register	void
enum	return	volatile
extern	short	while

__asm	__finally	__segment
__based	__fortran	__segname
__cdecl	__huge	__self

__emit	__interrupt	__stdcall
__except	__loadds	__syscall
__export	__near	__try
__far	__pascal	
__fastcall	__saveregs	

注释：上面所列的扩展关键字前面有两个下划线。不过为以后的可兼容性，这些关键字的单下划线版本也得到支持，除非指定 / Za 编译选项。

注意关键字 near、far、huge、cdece、fortran、pascal 和 interrupt 的使用方法。

除上面所示的 C++关键字外，C++定义表 1.1 中的名字为宏。用 ifdef 或 ifdefined 预处理指令可测试某些宏。

1.5 标点符号

C++中的标点符号对编译程序具有语法和语义含义，但不指定产生值的任何操作。有些标点符号也可是 C++算符，或对预处理程序有意义。标点符可是：

```
! % ^ & * ( ) - + = { } | ~
[ ] \ ; ' : " < > ? , . / #
```

标点符号必须在翻译阶段 4 后成对出现(有关详情见附录 A"翻译阶段")。

1.6 算 符

算符指定对三种操作数执行求值操作：一个操作数(一目算符)、二个操作数(二目算符)和三个操作数(三目算符)。

C++ 包括所有 C 算符，并添加若干新算符。如下语法先列举唯一于 C++的算符，然后列举 C 和 C++共享的算符。

C++算符

.*	::	delete
->*	new	

C / C++算符

=+	>	(
=%	<<	)
=/	>>	[
=*	%	]
=<<	=<	++
=>>	=>	<-
=-	<	.

--	==	&=
&	!=	^=
*	^	|=
+	|	,
-	&&	#
~	||	##
!	?	:>
sizeof	:	
/	=	

算符遵循严格的优先级规则。优先级规则定义含算符的表达式的求值顺序。算符要么与左边的表达式结合，要么与右边的表达式结合这称为结合性。表达式 1.2 说明了 C++算符的优先级和结合性(从最高级到最低级排列)。

表 1.2 C++算符优先级，语法和结合性

算符	含义	语法	结合性
==	Equality	*equality-expression* == *relational-expression*	Left to right
::	Scope resolution	*class-name* :: *name*	None
::	Global	:: *name*	None
[ ]	Array subscript	*postfix-expression* [$expression_{opt}$]	Left to right
( )	Function call	*postfix-expression* ($expression\text{-}list_{opt}$)	Left to right
( )	Conversion	*simple-type-name* ($expression\text{-}list_{opt}$)	None
.	Member selection (object)	*postfix-expression* . *name*	Left to right
->	Member selection (pointer)	*postfix-expression* -> *name*	Left to right
++	Postfix increment	*postfix-expression* ++	None
--	Postfix decrement	*postfix-expression* --	None
new	Allocate object	$::_{opt}$ **new** $placement_{opt}$ *new-type-name* $new\text{-}initializer_{opt}$	None
		$::_{opt}$ **new** $placement_{opt}$ (*type-name*) $new\text{-}initializer_{opt}$	None
delete	Deallocate object	$::_{opt}$ **delete** *cast-expression*	None
delete[ ]		$::_{opt}$ **delete** [ ] *cast-expression*	None
++	Prefix increment	++ *unary-expression*	None
--	Prefix decrement	-- *unary-expression*	None
*	Dereference	* *cast-expression*	None

&	Address-of	& *cast-expression*	None
+	Unary plus	+ *cast-expression*	None
–	Arithmetic negation (unary)	– *cast-expression*	None
!	Logical NOT	! *cast-expression*	None
~	Bitwise complement	~ *cast-expression*	None
:>	Base operator	*base-expression* :> *expression*	None
sizeof	Size of object	**sizeof** *unary-expression*	None
sizeof ( )	Size of type	**sizeof**(*type-name*)	None
(*type*)	Type cast (conversion)	(*type-name*) *cast-expression*	Right to left
.*	Apply pointer to class member (objects)	*pm-expression* .* *cast-expression*	Left to right
–>*	Dereference pointer to class member	*pm-expression* –>* *cast-expression*	Left to right
*	Multiplication	*multiplicative-expression* * *pm-expression*	Left to right
/	Division	*multiplicative-expression* / *pm-expression*	Left to right
%	Remainder	*multiplicative-expression* %	Left to right
+	Addition	*additive-expression* + *multiplicative-expression*	Left to right
–	Subtraction	*additive-expression* – *multiplicative-expression*	Left to right
<<	Left shift	*shift-expression* << *additive-expression*	Left to right
>>	Right shift	*shift-expression* >> *additive-expression*	Left to right
<	Less than	*relational-expression* < *shift-expression*	Left to right
>	Greater than	*relational-expression* > *shift-expression*	Left to right
<=	Less than or equal to	*relational-expression* <= *shift-expression*	Left to right
>=	Greater than or equal to	*relational-expression* >= *shift-expression*	Left to right
!=	Inequality	*equality-expression* != *relational-expression*	Left to right

&	Bitwise AND	*and-expression* & *equality-expression*	Left to right
^	Bitwise exclusive OR	*exclusive-or-expression* ^ *and-expression*	Left to right
|	Bitwise OR	*inclusive-or-expression* | *exclusive-or-expression*	Left to right
&&	Logical AND	*logical-and-expression* && *inclusive-or-expression*	Left to right
||	Logical OR	*logical-or-expression* || *logical-and-expression*	Left to right
e1?*e2*:*e3*	Conditional	*logical-or-expression* ? *expression* : *conditional-expression*	Right to left
=	Assignment	*unary-expression* = *assignment-expression*	Right to left
*=	Multiplication assignment	*unary-expression* *= *assignment-expression*	Right to left
/=	Division assignment	*unary-expression* /= *assignment-expression*	Right to left
%=	Modulus assignment	*unary-expression* %= *assignment-expression*	Right to left
+=	Addition assignment	*unary-expression* += *assignment-expression*	Right to left
-=	Subtraction assignment	*unary-expression* -= *assignment-expression*	Right to left
<<=	Left-shift assignment	*unary-expression* <<= *assignment-expression*	Right to left
>>=	Right-shift assignment	*unary-expression* >>= *assignment-expression*	Right to left
&=	Bitwise AND assignment	*unary-expression* &= *assignment-expression*	Right to left
|=	Bitwise inclusive OR assignment	*unary-expression* |= *assignment-expression*	Right to left
^=	Bitwise exclusive OR assignment	*unary-expression* ^= *assignment-expression*	Right to left
,	Comma	*expression*, *assignment-expression*	Left to right

[ ], ( )和?: 算符(分别为数值下标，函数调用和条件算符)仅可成对使用。不过这些算符可用表达式分隔开(有关详情见第四章"表达式")。##和##算符仅可出现在#define 预处理命令中。

1.7 文 字

程序的不变元素称为"文字"或"常量"。术语"文字"和"常量"在此可互用。文字主要分为四类：整型文字，字符文字，浮点文字和串文字。

1.7.1 整型常量

整型常量就是没有小数部分或指数部分的常量数据元素。它们始终以数字开头。整型常数可是十进制形式，八进制形式或十六进制形式，其类型可是带符号类型，无符号类型，长类型或短类型。其语法为：

整型常量：{十进制常整型后缀*opt*
八进制常整型后缀*opt*
十六进制常整型后缀*opt*
'C – char – sequence'

十进制常量：{非零数字
十进制常量数字

八进制常量：{0
八进制常量八进制数字

十六进制常量：{*0x* 十六进制数字
0X 十六进制数字
十六进制常量十六进制数字

非零数字：1 2 3 4 5 6 7 8 9

八进制数字：0 1 2 3 4 5 6 7

十六进制数字：{0 1 2 3 4 5 6 7 8 9
a b c d e f
A B C D E F

整型后缀：{无符号后缀长后缀*opt*
长后缀无符号后缀*opt*

无符号后缀：*u* *U*

长后缀：*L*

要用八进制或十六进制指定整型常量，使用表示基数的前缀。要指定给定整数类型的整型常量，使用表示类型的后缀。

要指定十进制常量。以非零数字开头 。例如：

```
int i = 157;   // Decimal constant
int j = 0198;  // Not a decimal number; erroneous octal constant
int k = 0365;  // Leading zero specifies octal constant, not decimal
```

要指定八进制常量，以 0 开，后面跟 0～7 之间的数字序列。在八进制常量中不得出