第 3 章　使用变量和常量

程序需要一种方式来存储其使用或创建的数据，以便在后面的程序执行期间能够使用它们。变量和常量提供各种表示、存储和操纵数据的方式。

本章将学习：

- 如何声明和定义变量与常量？
- 如何给变量赋值以及操纵这些值？
- 如何将变量的值写入到屏幕？

3.1　什么是变量

在 C++中，变量是存储信息的空间。变量是计算内存中的一个位置，可以在其中存储值或检索其中的值。

变量用于临时存储，退出程序或关机后，变量中的信息将丢失。永久存储则不同，变量的值被永久地存储到数据库或磁盘文件中。第 16 章将讨论存储到磁盘文件中。

3.1.1　将数据存储在内存中

可将计算机内存看作是一系列文件柜，每个文件柜由许多排成一列的小格子组成。每个文件柜（内存单元）都按顺序进行编号。这些编号被称为内存地址。变量通常预留一个或多个文件柜，可以在其中存储一个值。

变量名（如 myVariable）是贴在文件柜上的标签，这样无需知道变量的地址，就可方便地找到变量。图 3.1 说明了这一点。从中可知，变量 myVariable 从内存地址 103 开始。根据 myVariable 的大小，它可能占用一个或多个内存地址。

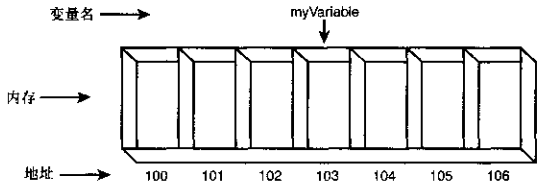

图 3.1　内存示意图

注意：RAM（Random Access Memory）即随机存取存储器。运行程序时，将把磁盘文件中的程序加载到 RAM 中。所有变量都是在 RAM 中创建的。程序员谈到内存时，通常指的是 RAM。

3.1.2　预留内存

在 C++中定义变量时，必须告诉编译器该变量的类型：整数、浮点数、字符型等。这种信息告诉编译器

预留多大空间以及你将在变量中存储什么样的值；它还让编译器能够在你将类型不对的值存储到变量中时发出警告或错误消息，编程语言的这种特征被称为类型检查（strong typing）。

每个文件格的大小都是 1 字节。如果你创建的变量占 4 字节，则需要 4 字节内存，也就是 4 个文件格。变量类型（如整型）告诉编译器给变量预留多大内存（多少个文件格）。

有时候，程序员必须理解“位”和“字节”的含义；毕竟这些是基本的存储单位。虽然计算机程序在剔除这些细节方面越来越强，但了解数据是如何存储的仍有帮助。有关二进制数学中底层概念的简介，请参阅附录 A。

注意：如果数学使你厌烦，就不要学习附录 A；读者并非真正需要这方面的知识。实际上，虽然熟悉逻辑和理性思维很重要，但程序员不再需要是数学家。

3.1.3　整型变量的大小

在任何计算机中，每种变量都占据一定的内存。也就是说，整数在一台机器上可能是 2 字节，而在另一台机器上可能是 4 字节，但对任何一台机器而言，这个值是固定的，不随时间而变化。

单个字符（包括字母、数字和符号）用 char 变量存储，这种变量通常为 1 字节。

对于较小的整数，可以使用 short 变量来存储。在大多数计算机上，short 变量为 2 字节，long 变量为 4 字节，而 int 变量（没有关键字 short 或 long）通常为 2 或 4 字节。

读者可能认为，语言规定了每种类型的长度，但 C++没有这样做，它只这样规定：short 的长度不能超过 int，而 int 的长度不能超过 long。

也就是说，在你使用的计算机上，short 可能为 2 字节，int 和 long 都是 4 字节。

int 的长度由处理器（16 位、32 位或 64 位）和编译器决定。在使用现代编译器和 Intel 奔腾处理器的 32 位计算机上，int 为 4 字节。

注意：创建程序时，不要将任何一种类型占用的内存量做出假设。

请编译并运行程序清单 3.1，它将指出在你的计算机上这些类型的长度。

程序清单 3.1　确定在你的计算机上变量类型的长度

```
#include <iostream>

int main()
{
   using std::cout;

   cout << "The size of an int is:\t\t"
      << sizeof(int)    << " bytes.\n";
   cout << "The size of a short int is:\t"
      << sizeof(short)  << " bytes.\n";
   cout << "The size of a long int is:\t"
      << sizeof(long)   << " bytes.\n";
   cout << "The size of a char is:\t\t"
      << sizeof(char)   << " bytes.\n";
   cout << "The size of a float is:\t\t"
      << sizeof(float)  << " bytes.\n";
   cout << "The size of a double is:\t"
      << sizeof(double) << " bytes.\n";
   cout << "The size of a bool is:\t"
      << sizeof(bool)   << " bytes.\n";

   return 0;
```

```
23: }
```

输出：

```
The size of an int is:          4 bytes.
The size of a short int is:     2 bytes.
The size of a long int is:      4 bytes.
The size of a char is:          1 bytes.
The size of a float is:         4 bytes.
The size of a double is:        8 bytes.
The size of a bool is:          1 bytes.
```

注意：在你的计算机上，显示的字节数可能与上面不同。

读者应该非常熟悉程序清单 3.1 中的大部分内容。其中有些代码行被分成多行，以适合本书的版面，例如，第 7 行和 8 行实际上应该为一行。编译器忽略空白（空格、制表符和换行符），因此可以将这两行看成一行。这也是在很多代码行末尾需要（;）的原因。

该程序中的一个新特性是，在第 7~20 行使用了运算符 sizeof。sizeof 的用法类似于函数，被调用时，它指出作为参数传递给它的类型的长度。例如，第 8 行将关键字 int 传递给了 sizeof。本章后面将指出，int 用于描述标准整型变量。在使用奔腾 4 和 Windows XP 的计算机上使用 sizeof 时，它将指出 int 的长度为 4 字节，这正好与 long int 的长度相同。

程序清单 3.1 中的其他行显示了其他数据类型的长度。下面用几分钟的时间简单介绍一下这些数据类型可以存储的值及其之间的差异。

signed 和 unsigned

所有整型类型都有两种变体：signed 和 unsigned。有时候，要求整型变量能够存储负数，有时候则不要求。没有使用关键字 unsigned 声明的整型变量都被视为无符号的，这种变量可以为正，也可以为负；而 unsigned 整型变量只能为正。

signed 和 unsigned 整型变量占用的内存空间相同，而 signed 整型变量的部分存储空间被用于存储指出该变量为正还是负的信息，因此 unsigned 整型变量能够存储的最大值为 signed 整型变量能够存储的最大正数的两倍。

例如，如果 short 变量占用 2 字节，则 unsigned short 变量的取值范围为 0~65 535，而 signed short 变量的取值范围内一半为正数，即它能够存储的最大正数为 32 767。然而，signed short 变量也能够存储负数，因此其取值范围为–32 768~32 767。

有关运算符优先级的详情，请参阅附录 C。

3.1.4 基本变量类型

C++内置了几种变量类型；它们分为整型变量、浮点变量和字符变量。

浮点变量的值可以表示为小数，也就是说它们是实数。字符变量只占 1 字节，通常用于存储 ASCII 字符集和扩展 ASCII 字符集中的 256 个字母和符号。

注意：ASCII 字符集是计算机使用的标准化字符集。ASCII 是 America Standard Code for Information Interchange（美国信息交换标准码）的缩写。几乎所有的计算机操作系统都支持 ASCII 码，虽然很多操作系统还支持其他的国际字符集。

表 3.1 描述了 C++程序中使用的变量类型，其中列出了变量类型、占有的内存量以及能够存储的值。变量类型能够存储的值是由其长度决定的，请查看程序清单 3.1 的输出，了解在你的计算机上，变量类型的长度是否与此相同。除非读者的计算机使用的是 64 位处理器，否则变量类型的长度很可能与此相同。

表 3.1　　变量类型

类　型	长　度	值
bool	1 字节	true/false
unsigned short int	2 字节	0 ~65 535
short int	2 字节	-32 768 ~32 767
unsigned long int	4 字节	0 ~4 294 967 295
long int	4 字节	–2 147 483 648~2 147 483 647
int（16 位）	2 字节	–32 768 ~32 767
int（32 位）	4 字节	-2 147 483 648 ~2 147 483 647
unsigned int （16 位）	2 字节	0~ 65 535
unsigned int （32 位）	4 字节	0~4 294 967 295
char	1 字节	256 个字符
float	4 字节	1 2e-38~3.4e38
double	8 字节	2 2e-308~1.8e308

注意：根据你使用的计算机和编译器，变量的长度可能与表 3.1 所示不同。如果你的计算机的输出和程序清单 3.1 相同，表 3.1 将适用于你的编译器；否则应查阅编译器手册，了解各种类型的变量能存储的值。

3.2　定义变量

至此，读者看到过很多被创建和使用的变量，下面介绍如何创建变量。

要创建或定义变量，可声明其类型，加上一个或多个空格，然后指出变量名，再加上一个分号。变量名可以是任何字母组合，但其中不能有空格。x、J23qrsnf 和 myAge 都是合法的变量名。好的变量名指出了其用途：使用好的变量名可使程序流程更容易理解。下面的语句定义了一个名为 myAge 的整型变量：

```
int myAge;
```

注意：声明变量时，将为该变量分配内存。此时相应内存中的值就是该变量的值。稍后将介绍如何给内存赋新值。

一种通用的编程惯例是，应避免使用像 J23qrsnf 这样难懂的变量名，也尽量不要使用单个字母（如 x 或 i）作为变量名。尽可能使用有意义的变量名，如 myAge 或 howMany。当你三周后再次阅读程序时，将很容易知道当时编写代码行时使用的变量名的含义。

请根据前几行代码猜测下面几个程序是做什么的。

范例 1：

```
int main()
{
   unsigned short x;
   unsigned short y;
   unsigned short z;
   z = x * y;
   return 0;
}
```

范例 2：

```
int main()
{
```

```
    unsigned short Width;
    unsigned short Length;
    unsigned short Area;
    Area = Width * Length;
    return 0;
}
```

注意：如果编译这两个程序，编译器将发出警告，指出有些值没有初始化。稍后你将知道如何解决这种问题。

显然，第 2 个程序的目的更易猜出。输入较长的变量名带来的不便，因第二个程序更容易理解和维护得到了补偿。

3.2.1 区分大小写

C++区分大小写，换句话说，大小写字母是不同的。例如，age、Age 和 AGE 是不同的变量。

注意：有些编译器允许关闭大小写区分功能。请不要这样做，因为这样做，使用其他编译器时程序将不能通过编译，同时其他 C++程序员将对你的代码感到非常迷惑。

3.2.2 命名规则

关于如何给变量命名的规则有很多，虽然采用哪种方法关系不大，在整个程序中保持一致很重要。命名规则不一致将使其他程序员阅读你的代码时感到迷惑。

很多程序员喜欢全部用小写字母表示变量名。如果变量名由两个单词组成（如 my car），有两种流行的表示法：my_car 和 myCar。后者被称为驼峰式表示法，因为其中的大写字母很像驼峰。

有些人认为下划线表示法（如 my_car）易读，但另一些人则尽量避免使用下划线，因为下划线更难输入。本书使用驼峰式表示法：变量名中第 2 个及以后的单词的首字母均大写，如 myCar、theQuickBrownFox 等。

许多高级程序员使用匈牙利表示法。匈牙利表示法的基本思想是，变量名以一组表示其类型的字符打头。例如，整型变量可能以小写字母 i 打头，而 long 变量可能以小写字母 l 打头。有关指示 C++中其他结构（如常量、全局变量、指针等）的表示法，将在后面介绍。

注意：之所以称为匈牙利表示法，是因为其发明者微软公司的 Charles Simonyi 是匈牙利人。可以在 http://www.strangecreations.com//library/c/naming.txt 找到他最初的论文。

微软近来已摒弃匈牙利表示法，其 C#设计建议中强烈建议不要使用匈牙利表示法。有关 C#的理由也适用于 C++。

3.2.3 关键字

有些字被 C++保留，不能用来作变量名。对 C++编译器来说，这些关键字有特殊含义。关键字包括 if、while、for、main 等。表 3.2 和附录 B 列出了 C++定义的关键字。你的编译器可能还保留了其他字，有关完整的关键字列表，请参阅编译器手册。

表 3.2　　C++关键字

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename

续表

class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	
另外，下列字被保留			
And	bitor	not_eq	xor
and_eq	compl	or	xor_eq
bitand	not	or_eq	

应该：

定义变量时先给出类型，然后是变量名。

使用有意义的变量名。

别忘了 C++区分大小写。

务必了解各种变量在内存中占用的字节数及其能够存储哪些值。

不应该：

不要将 C++关键字用作变量名。

不要对存储变量需要多少字节内存进行假设。

不要将负数赋给 unsigned 变量。

3.3　一次创建多个变量

可以在一条语句中创建多个类型相同的变量，方法是先指出变量类型，然后指定变量名，并用逗号将变量名分开。例如：

```
unsigned int myAge, myWeight;    // two unsigned int variables
long int area, width, length;    // three long integers
```

从中可知，变量 myAge 和 myWeight 都被声明为 unsigned int 变量。第 2 行声明了 3 个名为 area、width 和 length 的 long 变量。类型 long 被用于所有变量，因此，不能在同一条语句中定义不同类型的变量。

3.4　给变量赋值

可以使用赋值运算符（=）给变量赋值。要将 5 赋给变量 width，可以这样做：

```
unsigned short width;
width = 5;
```

注意： long 是 long int 的简写；short 是 short int 的简写。

可以将创建变量和给它赋值的步骤合而为一。例如，对于变量 width，将这两个步骤合而为一的代码如下：

```
unsigned short width = 5;
```

初始化与前面的赋值极其相似，对于像 width 这样的整型变量，两者的差别很小。后面介绍 const 时，读者将发现，有些变量必须初始化，因为以后不能给它们赋值。

可以一次定义多个变量，也可以在创建时初始化多个变量。例如，下面的代码创建两个 long 变量，并初始化它们：

```
long width = 5, length = 7;
```

上述代码将 long 变量 width 和 length 分别初始化为 5 和 7。你甚至可以将定义和初始化混合起来：

```
int myAge = 39, yourAge, hisAge = 40;
```

上述代码创建 3 个 int 变量，并初始化第 1 个（myAge）和第 3 个（hisAge）变量。

程序清单 3.2 是一个可以编译的完整程序，它计算矩形的面积并将结果打印到屏幕上。

程序清单 3.2　演示变量的用法

```
1:  // Demonstration of variables
2:  #include <iostream>
3:
4:  int main()
5:  {
6:     using std::cout;
7:     using std::endl;
8:
9:     unsigned short int Width = 5, Length;
10:    Length = 10;
11:
12:    // create an unsigned short and initialize with result
13:    // of multiplying Width by Length
14:    unsigned short int Area = (Width * Length);
15:
16:    cout << "Width:" << Width << endl;
17:    cout << "Length: " << Length << endl;
18:    cout << "Area: " << Area << endl;
19:    return 0;
20: }
```

输出：

```
Width:5
Length: 10
Area: 50
```

分析：

从上述程序清单可知，第 2 行的 include 语句包含 iostream 库，以便能够使用 cout。程序从第 4 行的 main() 函数开始。第 6 行和第 7 行指出 cout 和 endl 位于标准（std）名称空间中。

在第 9 行，定义了第一个变量。Width 被定义为 unsigned short 变量，并被初始化为 5。同时，还定义了另一个 unsigned short 变量 Length，但没有初始化。在第 10 行，将值 10 赋给了 Length。

在第 14 行，定义了 unsigned short 变量 Area，并将其初始化为 Width 和 Length 的乘积。在第 16～18 行，将各个变量的值打印到屏幕上。注意，特殊单词 endl 另起一行。

3.5　使用 typedef 来创建别名

不断地输入 unsigned short int，既繁琐又容易出错。C++允许你使用关键字 typedef（表示类型定义）为这

个短语创建一个别名。

实际上，这创建的是一个同义词，将其与第 6 章将介绍的创建新类型区分开来尤关重要。要创建别名，可使用关键字 typedef，后面跟现有类型和新名称，并以分号结束。例如，下面的语句创建新名称 USHORT，你可以在任何本应使用 unsigned short int 的地方使用它：

```
typedef unsigned short int USHORT;
```

程序清单 3.3 与程序清单 3.2 相同，但使用了类型定义 USHORT，而不是 unsigned short int。

程序清单 3.3　演示 typedef

```
 1: // Demonstrates typedef keyword
 2: #include <iostream>
 3:
 4: typedef unsigned short int USHORT;   //typedef defined
 5:
 6: int main()
 7: {
 8:
 9:    using std::cout;
10:    using std::endl;
11:
12:    USHORT Width = 5;
13:    USHORT Length;
14:    Length = 10;
15:    USHORT Area  = Width * Length;
16:    cout << "Width:" << Width << endl;
17:    cout << "Length: "  << Length << endl;
18:    cout << "Area: " << Area <<endl;
19:    return 0;
20: }
```

输出：

```
Width:5
Length: 10
Area: 50
```

注意：星号（*）表示乘法运算。

分析：

在第 4 行，USHORT 被定义为 unsigned short int 的同义词。该程序与程序清单 3.2 极其相似，输出也相同。

3.6　何时使用 short 和 long

一个令 C++新手感到困惑的问题是，什么时候将变量声明为 long 类型，什么时候将其声明为 short 类型。规则非常简单：可能需要将比其类型能够存储的最大值还要大的值赋给变量时，应将该变量声明为更长的类型。

如表 3.1 所示，unsigned short 变量占用 2 字节，能够存储的最大值为 65 535；signed short 变量的取值范围由正数和负数两部分组成，因此能够存储的最大值为前者的一半。

虽然 unsigned long 变量可存储非常大的值（4 294 967 295），但这依然很有限。如果需要存储更大的值，必须使用 float 或 double 变量，但这样会降低精度。float 和 double 变量能够存储非常大的值，但在大多数计算机上，只有前 7 位或 9 位有效，这意味着其后位将被四舍五入。

变量越短，占用的越少。现在，内存已很便宜，而时间却很宝贵。请放心使用 int，在你的计算机上，它可能占用 4 字节。

3.6.1 unsigned 整型变量的回绕

unsigned long 变量能够存储的最大值有限，但这通常不是问题。如果数据确实超过了允许的最大值，将出现什么情况呢？

当 unsigned 整型变量达到其最大值时将回绕，就像汽车里程表一样。程序清单 3.4 演示了将过大的值赋给 short 变量时出现的情况。

程序清单 3.4 将过大的值赋给 unsignde short 变量

```
1:  #include <iostream>
2:  int main()
3:  {
4:     using std::cout;
5:     using std::endl;
6:
7:     unsigned short int smallNumber;
8:     smallNumber = 65535;
9:     cout << "small number:" << smallNumber << endl;
10:    smallNumber++;
11:    cout << "small number:" << smallNumber << endl;
12:    smallNumber++;
13:    cout << "small number:" << smallNumber << endl;
14:    return 0;
15: }
```

输出：

```
small number:65535
small number:0
small number:1
```

分析：

在第 7 行，smallNumber 被声明为 unsignde short int 变量，在笔者使用的运行 Windows XP 的奔腾 4 计算机上，这种变量占用 2 字节，取值范围为 0～65 535。在第 8 行，将可存储的最大值赋给了 smallNumber，第 9 行打印它。

在第 10 行，smallNumber 被递增，即将其加 1。递增运算符为++，就像 C++表明它是从 C 语言增补而来的。这样，smallNumber 的值将为 65 536，然而，unsigned short 变量不能存储大于 65 535 的值，因此这个值回绕为 0，第 11 行打印了它。

在第 12 行，再次对 smallNumber 进行了递增，然后打印新的值 1。

3.6.2 signed 整型变量的回绕

signed 整型变量与 unsigned 整型变量的区别在于，在它能够存储的值中，有一半为负。假设你要模拟的不是传统的汽车里程表，而是如图 3.2 所示的钟表，其中的数字按顺时针递增，按逆时针递减，它们在钟面底部（即 6 点钟处）交叉。

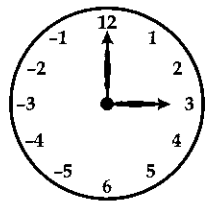

图 3.2　如果钟表使用有符号数字

从 0 开始数，下一个数为 1（顺时针方向）或–1（逆时针方向）。数完正数后，接下来就是最大的负数，最终数会到 0。程序清单 3.5 说明了当 short 整型变量为最大允许正数时再加 1 后发生的情况。

程序清单 3.5　将过大的值赋给 signed short 变量

```
1:  #include <iostream>
2:  int main()
3:  {
4:     short int smallNumber;
5:     smallNumber = 32767;
6:     std::cout << "small number:" << smallNumber << std::endl;
7:     smallNumber++;
8:     std::cout << "small number:" << smallNumber << std::endl;
9:     smallNumber++;
10:    std::cout << "small number:" << smallNumber << std::endl;
11:    return 0;
12: }
```

输出：

```
small number:32767
small number:-32768
small number:-32767
```

分析：

在第 4 行，smallNumber 被定义 signed short 变量（如果没有指明为无符号的，整型变量被视为有符号的）。该程序与前一个程序几乎相同，但输出完全不同。要完全理解其输出，必须知道有符号数如何被表示为两字节整型中的位。

然而，有符号整型的回绕与无符号整型相同，从最大正值回绕为最小负值。

3.7　使用字符

字符变量（类型为 char）通常只占 1 字节，能够存储 256 个值（见附录 C）。char 变量可被解释为 0～255 的数或 ASCII 字符集中的成员。ASCII 字符集和对应的 ISO 字符集是一种将字母、数字、标点符号进行编码的方式。

注意：计算机没有字母、标点符号和句子的概念，只能理解数字。实际上，计算机真正能识别的只是线路结点处是否有足够的电压；这两种状态用符号 1 和 0 表示。通过一系列 0 和 1 的组合，计算机能生成可被解释为数字的模式，而数字又可进一步被解释为字母和标点符号。

小写字母 a 的 ASCII 码为 97。所有大小写字母、数字、标点符号的 ASCII 码都是 1～128 的值。另外的

128 个标记和符号被留给计算机制造商使用，虽然 IBM 的扩展字符集在某种程度上已成为标准。

3.7.1 字符和数字

将一个字符（如字母 a）赋给 char 变量时，该变量实际存储的是一个 0～255 的值。然而，编译器知道如何在字符（用单引号括起的字母、数字或标点符号）和 ASCII 码之间进行转换。

这种数值与字母之间的关系是任意的，也就是说，小写字母 a 并非一定要对应 97。只要所有设备（键盘、编译器、显示器）就数值与字符之间的关系达成一致，就不会出现问题。然而，需要指出的是，数值 5 与字符 5 之间有天壤之别。后者对应的 ASCII 码为 53，就像 a 的 ASCII 码为 97 一样。程序清单 3.6 说明了这一点。

程序清单 3.6 使用数字来打印字符

```
1:  #include <iostream>
2:  int main()
3:  {
4:     for (int i = 32; i<128; i++)
5:        std::cout << (char) i;
6:     return 0;
7:  }
```

输出：

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcde-
fghijklmno
pqrstuvwxyz{|}~?
```

分析：

这个简单程序打印 ASCII 码为 32～127 的字符。

它在第 4 行使用一个整型变量 i 来完成这项任务。在第 5 行，强制将变量 i 的值显示为字符。

也可以使用字符变量，如程序清单 3.7 所示，其输出相同。

程序清单 3.7 使用数字来打印字符

```
1:  #include <iostream>
2:  int main()
2:  {
4:     for (unsigned char i = 32; i<128; i++)
5:        std::cout << i;
6:     return 0;
7:  }
```

正如读者看到的，第 4 行使用了一个 unsigned char 变量。由于使用的是字符变量而不是数值变量，因此第 5 行的 cout 知道应该显示字符值。

3.7.2 特殊打印字符

C++编译器能够识别一些特殊的格式化字符。表 3.3 列出了最常用的格式化字符。要在代码中使用它们，可输入反斜杠（被称为转义字符）和相应的字符。例如，要在代码中加入制表符，可依次输入左单引号、反斜杠、字母 t、右单引号：

```
char tabCharacter = '\t';
```

上述代码声明了一个 char 变量（tabCharacter），并将其初始化为\t——制表符。特殊打印字符用于将输出发送到屏幕、文件或其他输出设备。

转义字符（\）改变了其后面的字母的含义。例如，通常情况下，n 表示字母 n，但在前面加上转义字符后，表示换行。

表 3.3　　转 义 字 符

字　符	含　义
\a	响铃
\b	退格
\f	换页
\n	换行
\r	回车
\t	制表符
\v	垂直制表符
\'	单引号
\"	双引号
\?	问号
\\	反斜杠
\000	八进制表示
\xhhh	十六进制表示

3.8　常　　量

与变量一样，常量也是数据的存储位置；与变量不同的是，顾名思义，常量的值不能修改。创建常量时必须初始化，且以后不能给它赋值。

C++有两种常量：字面常量和符号常量。

3.8.1　字面常量

字面常量指直接输入到程序中的值。例如，在下面的代码中：

```
int myAge = 39;
```

其中 myAge 是一个 int 变量，而 39 是一个字面常量。不能给 39 赋值，其值也不能修改。

3.8.2　符号常量

符号常量指用名称表示的常量，就像表示变量一样。然而，与变量不同的是，常量一旦被初始化，其值就不能改变。

如果程序中有两个整型变量：students 和 classes，如果每班有 15 名学生，给定班级数，将可以求出学生数：

```
students = classes*15;
```

在这个例子中，15 是一个字面常量。如果用一个符号常量代替它，程序将更容易理解和维护：

```
students = classes * studentsPerClass
```

如果以后想修改每班的人数，只需在定义常量 studentsPerClass 的地方进行修改，而不必逐个修改。

在 C++中，有两种声明符号常量的方法。老式（传统）方法是使用预处理器编译指令#define，这种方法现已摒弃。另一种也是更合适的方法是，使用关键字 const 来创建它们。

1. 使用#define 定义常量

由于很多现有的程序使用预处理器编译指令#define，因此了解其用法很重要。要以这种过时的方式定义常量，可以这样做：

```
#define studentsPerClass 15
```

请注意，这里没有指定 studentsPerClass 的类型（int、char 等）。预处理器进行简单的文本替换。对这个例子而言，每当预处理器看到 studentPerClass 时，都将它替换为文本 15。

由于预处理器在编译器之前运行，因此编译器只能看到 15，而看不到常量 studentsPerClass。

警告：虽然#define 看似使用起来非常容易，但应避免使用它，因为 C++标准指出它已过时。

2. 用 const 定义常量

虽然#define 可行，但 C++中有一种更好的定义常量的方法：

```
const unsigned short int studentsPerClass = 15;
```

这个例子也声明了一个名为 studentPerClass 的符号常量，但这次 studentPerClass 被声明为 unsigned short int 类型。

这种声明常量的方法在使程序易于维护和防止错误方面有若干优点。最大的不同在于，该常量有类型，这使得编译器能够根据类型确保它被正确使用。

注意：在程序运行过程中，常量不能被修改。例如，要修改 studentPerClass 的值，应修改源代码并重新编译。

应该：

谨防整型变量的值超出允许的范围，否则将回绕得到错误的值。

务必使用能反映变量用途的变量名。

不应该：

不要将 C++关键字用作变量名。

不要使用预处理器编译指令#define 来声明常量，而应使用 const。

3.9 枚举常量

枚举型常量使你能够创建新类型，然后定义新类型变量，将这些变量的取值限定为一组可能值。例如，可以创建一个枚举来存储颜色。具体地说，可以将 COLOR 声明为枚举，然而指定 COLOR 的 5 个可能取值：RED、BLUE、GREEN、WHITE 和 BLACK。

创建枚举型常量的语法是：先输入关键字 enum，接着为新的类型名、左大括号、用逗号分隔的所有合法值、右大括号和分号，如下例所示：

```
enum COLOR {RED, BLUE, GREEN, WHITE, BLACK};
```

该语句执行两项任务：

（1）将 COLOR 指定为枚举的名称，即它将是一种新类型。

（2）使 RED 成为一个符号常量，其值为 0；BLUE 也为符号常量，其值为 1；GREEN 为符号常量，值为 2；依此类推。

每个枚举型常量都有一个整数值。如不特别指定，第一个常量的值为 0，其余常量依次递增。然而，可将其中任何一个常量初始化为特定的值，这样，后面未被初始化的常量将在这个基础依次递增。因此，如果有下面的代码：

```
enum Color {RED=100, BLUE, GREEN=500, WHITE, BLACK=700};
```

则 RED 的值将为 100，BLUE 的值为 101，GREEN 的值为 500，WHITE 的值为 501，BLACK 的值为 700。

可以定义类型为 COLOR 的变量，但它们的值只能是某个枚举值（在这个例子中，为 RED、BLUE、

GREEN、WHITE 或 BLACK）。可以将其中的任何颜色值赋给 COLOR 变量。

需要注意的是，枚举变量的类型通常为 unsigned int，而枚举常量相当于整型常量。然而，使用颜色、星期几及类似的信息时，如果能够给这些值指定名称，将非常方便。程序清单 3.8 是一个使用枚举类型的程序。

程序清单 3.8　枚举型常量

```
1:  #include <iostream>
2:  int main()
3:  {
4:     enum Days { Sunday, Monday, Tuesday,
5:              Wednesday, Thursday, Friday, Saturday };
6:
7:     Days today;
8:     today = Monday;
9:
10:    if (today == Sunday || today == Saturday)
11:       std::cout << "\nGotta' love the weekends!\n";
12:    else
13:       std::cout << "\nBack to work.\n";
14:
15:    return 0;
16: }
```

输出：

```
Back to work.
```

分析：

在第 4 和 5 行，定义了枚举类型 Days，它有 7 个可能的取值。每个值对应一个整数，从 0 开始计算，因此 Monday 的值为 1（Sunday 的值为 0）。

在第 7 行，创建了一个类型为 Days 的变量，也就是说，这个变量将可以取第 4 和 5 行定义的枚举常量之一。第 8 行将值 Monday 赋给该变量；第 10 行对这个变量的值进行检测。

可以使用一系列整型常量而不是枚举类型，如程序清单 3.9 所示。

程序清单 3.9　使用整型常量

```
1:  #include <iostream>
2:  int main()
3:  {
4:     const int Sunday = 0;
5:     const int Monday = 1;
6:     const int Tuesday = 2;
7:     const int Wednesday = 3;
8:     const int Thursday = 4;
9:     const int Friday = 5;
10:    const int Saturday = 6;
11:
12:    int today;
13:    today = Monday;
14:
15:    if (today == Sunday || today == Saturday)
16:       std::cout << "\nGotta' love the weekends!\n";
17:    else
18:       std::cout << "\nBack to work.\n";
19:
```

```
20:    return 0;
21: }
```

输出：

```
Back to work.
```

警告：在这个程序中声明的很多常量没有被使用过，因此编译该程序清单时，编译器可能发出警告。

分析：

该程序清单的输出与程序清单 3.8 相同。其中每个常量（Sunday、Monday 等）都被显式地定义，且使用枚举类型 Days。枚举常量的优点一目了然：枚举类型 Days 的含义是非常清楚的。

3.10 小　　结

本章讨论了数值变量、字符变量和常量，C++使用它们在程序执行期间存储数据。数值变量可以是整型（char、int 和 long int）或浮点型（float、double 和 long double）。另外，数值变量还可以是有符号或无符号的。虽然各种变量类型的长度随计算机而异，但在给定的计算机上，类型决定了变量的长度。

使用变量前必须声明它，然后只能将声明类型的数据存储到该变量中。如果将过大的值赋给整型变量，将回绕，导致错误的结果。

本章还介绍了字面常量、符号常量和枚举常量。读者学习了两种声明符号常量的方法：使用#define 和使用关键字 const；然而，后者是一种更合适的方式。

3.11 问　与　答

问：既然 short int 可能回绕，为什么不总是使用 long 变量呢？

答：所有整型变量都可能由于空间不够而回绕，但对于 long 变量，需要将更大的值赋给它时才会回绕。例如，两字节的 unsigned shrot int 变量在赋给它的值大于 65 535 时回绕，而 4 字节的 unsigned short int 变量在赋给它的值大于 4 294 967 295 时回绕。然而，在大多数计算机上，long 变量占用的内存是 int 变量的 2 倍（即 4 字节而不是 2 字节），如果程序中有 100 个这样的变量，将多消耗 200 字节的 RAM。坦率地说，与过去相比，这已经不是什么大问题，因为现在的个人计算机即使没有数吉字节也有数兆字节内存。

问：如果将一个带小数点的数赋给整型变量而不是浮点变量，将出现什么结果？如下述代码所示：

```
int aNumber = 5.4;
```

答：好的编译器会发出警告，但这种赋值是完全合法的。这个值将被截短为整数。因此，将 5.4 赋给一个整型变量，该变量的值将为 5。然而，将丢失信息，如果再将这个变量的值赋给一个 float 变量，后者的值也将为 5。

问：为什么不用字面常量，为什么要自找麻烦地使用符号常量？

答：如果在程序的很多地方都使用了同一个值，则使用符号常量时，只需修改该常量的定义，就可以修改所有地方的这个值。另外，符号常量的含义明确，在程序中乘以 360 时不好理解，但乘以符号常量 degreesInACircle 将一目了然。

问：如果将负数赋给一个 unsigned 变量将出现什么结果？如下述代码所示：

```
unsigned int aPositiveNumber = -1;
```

答：好的编译器会发出警告，但这种赋值是合法的。负数将被解释为位模式，并赋给变量。然后该变量的值将被解释为无符号数。-1 的位模式为 1111111111111111（用十六进制表示为 0xFF），它将被解释为无符

号值 65 535。

问：如果不懂位模式、二进制算术和十六进制，还能使用 C++吗？

答：能，但不如了解这些主题时有效。在无需了解计算机的工作原理方面，C++做得没有有些语言那么好，但这实际上是件好事，因为它提供了大量其他语言没有的功能。和任何功能强大的工具一样，要充分发挥 C++的潜力，必须了解其工作原理。使用 C++编程的程序员如果不了解二进制，将经常对结果感到迷惑。

3.12　作　　业

作业包括测验和练习，前者帮助加深读者对所学知识的理解，后者提供了使用新学知识的机会。请尽量先完成测验和练习题，然后再对照附录 D 中的答案，继续学习下一章之前，请务必弄懂这些答案。

3.12.1　测验

1. 整型变量和浮点变量之间有何不同？
2. unsigned short int 变量和 long int 变量之间有何不同？
3. 符号常量相对于字面常量有何优点？
4. 使用关键字 const 与使用#define 相比有何优点？
5. 什么决定了变量名是好还是坏？
6. 给定如下枚举类型，BLUE 的值是多少？

```
enum COLOR {WHITE, BLACK=100, RED, BLUE, GREEN=300};
```

7. 下列变量名哪些是好的？哪些是不好的？哪些是非法的？
   a. Age
   b. !ex
   c. R79J
   d. TotalIncome
   e. __Invalid

3.12.2　练习

1. 要存储下列信息，正确的变量类型是什么？
   a. 年龄
   b. 后院的面积
   c. 银河系中的星星数目
   d. 一月份的平均降水量
2. 给上述信息取一个好的变量名。
3. 声明一个常量来表示 π 等于 3.14159。
4. 声明一个 float 变量，并用常量 π 来初始化它。