# 8.3 默认参数

默认参数指的是当函数调用中省略了实参时自动使用的一个值。例如，如果将void wow(int n)设置成n有默认值为1，则函数调用wow()相当于wow(1)。这极大地提高了使用函数的灵活性。假设有一个名为left()的函数，它将字符串和n作为参数，并返回该字符串的前n个字符。更准确地说，该函数返回一个指针，该指针指向由原始字符串中被选中的部分组成的字符串。例如，函数调用left('theory”,3)将创建新字符串“the”，并返回一个指向该字符串的指针。现在假设第二个参数的默认值被设置为1，则函数调left(“theory”,3)仍像前面讲述的那样工作，3将覆盖默认值。但函数调用left(“theory”)不会出错，它认为第二个参数的值为1，并返回指向字符串“t”的指针。如果程序经常需要抽取一个字符组成的字符串，而偶尔需要抽取较长的字符串，则这种默认值将很有帮助。

如何设置默认值呢？必须通过函数原型。由于编译器通过查看原型来了解函数所使用的参数数目，因此函数原型也必须将可能的默认参数告知程序。方法是将值赋给原型中的参数。例如，left()的原型如下：

char \* left(const char \* str, int n=1);

您希望该函数返回一个新的字符串，因此将其类型设为char\*（指向char的指针）；您希望原始字符串保持不变，因此对第一个参数使用了const限定符；您希望n的默认值为1，因此将这个值赋给n。默认参数值是初始化值，因此上面的原型将n初始化为1。如果省略参数n，则它的值将为1；否则，传递的值将覆盖1。

对于带参数列表的函数，必须从右向左添加默认值。也就是说，要为某个参数设置默认值，则必须为它右边的所有参数提供默认值：

int harpo(int n , int m = 4 , int j = 5); // 有效

int chico(int n , int m = 6 , int j); // 无效

int groucho(int k = 1 , int m = 2 , int n = 3); // 有效

例如，harpo()原型允许调用该函数时提供1个、2个或3个参数：

beeps = harpo(2); // 与harpo(2,4,5)相同

beeps = harpo(1,8); // 与harpo(l,8)相同

beeps = harpo(8,7,6); // 没用默认参数

实参按从左到右的顺序依次被赋给相应的形参，而不能跳过任何参数。因此，下面的调用是不允许的：

beeps = harpo(3, ,8); // 无效，并没有将m设置为4

默认参数并非编程方面的重大突破，而只是提供了一种便捷的方式。在设计类时您将发现，通过使用默认参数，可以减少要定义的析构函数、方法以及方法重载的数量。

程序清单8.9使用了默认参数。请注意，只有原型指定了默认值。函数定义与没有默认参数时完全相同。

**程序清单8.9 left.cpp**

// left.cpp -- 使用一个默认参数的字符串处理函数

#include <iostream>

const int ArSize = 80;

char \* left(const char \* str, int n = 1);

int main()

{

using namespace std;

char sample[ArSize];

cout << "Enter a string:\n";

cin.get(sample,ArSize);

char \*ps = left(sample, 4);

cout << ps << endl;

delete [] ps; // 释放旧字符串

ps = left(sample);

cout << ps << endl;

delete [] ps; // 释放新字符串

cin.get();

cin.get();

return 0;

}

// 这个函数返回一个指向新字符串的指针，

// 这个字符串由字符串str的前n个字符组成

char \* left(const char \* str, int n)

{

if(n < 0)

n = 0;

char \* p = new char[n+1];

int i;

for (i = 0; i < n && str[i]; i++)

p[i] = str[i]; // 复制字符

while (i <= n)

p[i++] = '\0'; // 将字符串的剩余部分设置为'\0'

return p;

}

下面是该程序的运行情况：

Enter a string

forthcoming

fort

f

### 程序说明

该程序使用new创建一个新的字符串，以存储被选择的字符。一种可能出现的尴尬情况是，不合作的用户要求的字符数目可能为负。在这种情况下，函数将字符计数设置为0，并返回一个空字符串。另一种可能出现的尴尬情况是，不负责任的用户要求的字符数目可能多于字符串包含的字符数，为预防这种情况，函数使用了一个组合测试：

i < n && str[i]

i < n测试让循环复制了n个字符后终止。测试的第二部分——表达式str[i]，是要复制的字符的编码。遇到空值字符（其编码为0）后，循环将结束。这样，while循环将使字符串以空值字符结束，并将余下的空间（如果有的话）设置为空值字符。

另一种设置新字符串长度的方法是，将n设置为传递的值和字符串长度中较小的一个：

int len = strlen(str);

n = (n < len)? n : len // n和less中较小的一个

char \* p = new char[n+1];

这将确保new分配的空间不会多于存储字符串所需的空间。如果用户执行像left(“Hi!”,32767)这样的调用，则这种办法很有用。第一种方法将把“Hi”复制到由32767个字符组成的数组中，并将除前3个字符之外的所有字符都设置为空值字符；第二种方法将“Hi!”复制到由4个字符组成的数组中。，但由于添加了另外一个函数调用（strlen()），因此程序将更长，运行速度将降低，同时还必须包含头文件cstring（或string.h）。C程序员倾向于选择运行速度更快、更简洁的代码，因此需要程序员在正确使用函数方面承担更多责任。然而，C++的传统是更强调可靠性。毕竟，速度较慢但能正常运行的程序，要比运行速度虽快但无法正常运行的程序好。如果调用strIen()所需的时间很长，则可以让left()直接确定n和字符串长度哪个小。例如，当m的值等于n或到达字符串结尾时，下面的循环都将终止：

int m = 0;

while (m <= n && str[m] !=’\0’)

m++;

char \* p = new char[m+1];

// 在后面的代码中用m替换n

别忘了，在str[m]不是空值字符时，表达式str[m]!=’\0’的结果为true，否则为false。由于&&表达式中，非零值被转换为true，而零被转换为false，因此也可以这样编写这个while测试：

while (m<=n && str[m])