##**11.2** 为特定类型的数据命名

在为数据命名的时候，除了通常的考虑事项之外，为一些特定类型数据的命名还要求做出一些特殊的考虑。本节将讲述与循环变量、状态变量、临时变量、布尔变量、枚举类型和具名常量有关的考虑事项。

###为循环下标命名

循环是一种极其常见的计算机编程特征，为循环中的变量进行命名的原则也由此应运而生。i、j 和 k 这些名字都是约定俗成的：

Java 示例：简单的循环变量名

```
for ( i = firstItem; i < lastItem; i++ ){
    data[ i ]= 0;
}
```

如果一个变量要在循环之外使用，那么久应该为它取一个比 i、j 或者 k 更有意义的名字。举个例子，如果你在从文件中读取记录，并且需要记下所读记录是数量，那么类似于 recordCount 这样的名字就很适合：

Java 示例：描述性较好的循环变量名

```
recordCount = 0;
while ( moreSource() ){
    score[ recordCount ] = GetNextScore();
    recordCount++;
}

// lines using recordCount
...
```

如果循环不是只有几行，那么读者会很容易忘记 i 变量具有的含义，因此你最好给循环下标换一个更有意思的名字。由于代码会经常修改、扩充，或者复制到其他程序中去，因此，很多有经验的程序员索性不使用类似于 i 这样的名字。

导致循环变长的常见原因之一是出现循环的嵌套使用。如果你使用多个嵌套的循环，那么就应该给循环变量赋予更长的名字以提高可读性：

Java 示例：嵌套循环中的号循环变量名

```
for ( teamIndex = 0; teamIndex < teamCount; teamIndex++ ){
     for ( eventIndex = 0; eventIndex < eventCount[teamIndex]; eventIndex++ ){
          score[ teamIndex ][ eventIndex ] = 0;
     }
}
```

谨慎地为循环下标变量命名可以避免产生下标串话的常见问题：想用 j 的时候写了 i，想用 i 的时候却写了 j。同时，这也使得数据访问变得更加清晰：score[teamIndex][eventIndex] 要比 score[i][j] 给出的信息更多。

如果你一定要用 i、j 和 k，那么不要吧它们用于简单循环的循环下标之外的任何场合 —— 这种传统已经太深入人心了，一旦违背该原则，将这些变量用于其他用途就可能造成误解。要想避免出现这样的问题，最简单的方法就是想出一个比 i、j 和 k 更具描述性的名字来。

###为状态变量命名

状态变量用于描述你的程序的状态。下面给出它的命名原则。

为状态变量取一个比 **flag** 更好的名字 最好是把标记（flag）看做状态变量。标记的名字中不应该含有 flag，因为你从中丝毫看不出该标记是做什么的。为了清楚起见，标记应该用枚举类型、具名常量，或用作具名常量的全局变量来对其赋值，而且其值应该与上面这些量作比较。下面例子中标记的命名都很差：

C++ 示例：含义模糊的标记

```
if (flag)  ...
if (statusFlag & 0X0F)  ...
if (printFlag == 16)  ...
if (computeFlag == 0)   ...

flag = 0x1;
statusFlag = 0x80;
printFlag = 16;
computeFlag = 0;
```

像 statusFlag = 0x80 这样的语句是反映不出这段代码能做上面的，除非你亲自写了这段代码，或者有文档能告诉你 statusFlag 和 0x80 的含义。下面是作用相同但更为清晰的代码：

C++ 示例：更好地使用状态变量

```
if ( dateReady )  ...
if ( characterType & PRINTABLE_CHAR )  ...
if ( reportType == ReportType_Annual )  ...
if ( RecalcNeeded ==false )   ...

dateReady = true;
characterType = CONTROL_CHARCTER;
reportType = ReportType_Annual;
RecalcNeeded ==false;
```

显然，characterType = CONTROL_CHARCTER 要比 statusFlag = 0x80 更有意义。与之类似，条件判断语句 if (reportType == ReportType_Annual) 要比 if (printFlag == 16) 更为清晰。第二个例子说明你可以结合枚举类型和预定义的具名常量来使用这种方法。下面例子展示了如何使用具名常量和枚举类型来组织例子中的数值：

在 C++ 中声明状态变量

```
// values for CharacterType
const int = 0x01;
const int = 0x02;
const int = 0x04;
const int LINE_DRAW = 0x08;
const int PRINTABLE_CHAR = ( LETTER | DIGIT | PUNCTYATION | LINE_DRAW );
const int CONTROL_CHARACTER = 0x80;

// values for ReportType
enum ReportType{
    ReportType_Daily,
    ReportType_Monthly,
    ReportType_Quarterly,
    ReportType_Annual,
    ReportType_All
};
```

如果你放心自己需要猜测某段代码的含义的时候，就该考虑为变量重新命名。猜测谋杀案中谁是神秘凶手是可行的，但你没有必要去猜测代码，你应该能直接读懂它们。

###为临时变量命名

临时变量用于存储计算的中间结果，作为临时占位符，以及存储内存管理值。它们常被赋予 temp、 x 或者其他一些模糊且缺乏描述性的名字。通常，临时变量是一个信号，表明程序员还没有完全把问题弄清楚。而且，由于这些变量被正式地赋予了一种 “临时” 状态，因此程序员会倾向于其他变量更为随意地对待这些变量，从而增加了出错的可能。

警惕 “临时” 变量 临时性地保存一些值常常是很有必要的。但是无论从哪种角度看你程序中的大多数变量都是临时的。把其中几个称为临时的，可能表明你还没有弄清它们的实际用途。请考虑下面的示例：

`C++` 示例：不提供信息的 “临时” 变量名

```
// Compute roots of a quadratic equation.
// This assumes that ( b^2 - 4*a*c ) is positive.
temp = sqrt( b^2 - 4*a*c );
root[0] = ( -b + temp ) / ( 2*a );
root[1] = ( -b - temp ) / ( 2*a );
```

把表达式 sqrt(b^2 - 4*ac*) 的结果存储在一个变量里是很不错的，特别是当这一结果还会被随后两次用到的时候。但是名字 temp 却丝毫也没有反映该变量的功能。下面例子显示了一种更好的做法：

`C++` 示例：用真正的变量替代 “临时” 变量名

```
// Compute roots of a quadratic equation.
// This assumes that ( b^2 - 4*a*c ) is positive.
discriminant = sqrt( b^2 - 4*a*c );
root[0] = ( -b + discriminant ) / ( 2*a );
root[1] = ( -b - discriminant ) / ( 2*a );
```

就本质而言，这段代码与上面一段是完全相同的，但是它却通过使用了准确而且具有描述性的变量名（discriminant，判别式）而得到了改善。

###为布尔变量命名

下面是为布尔变量命名时时要遵循的几条原则。

谨记典型的布尔变量名。 下面是一些格外有用的布尔变量名。

- **done** 用 done 表示某件事已经完成。这一变量可用于表示循环结束或者一些其他的操作已完成。在事情完成之前把 done 设为 false，在事情完成之后把它设为 true。
- **error** 用 error 表示有错误发生。在错误发生之前把变量值设为 false，在错误已经发生时把它设为 true。
- **found** 用 found 表明某个值已经找到了。在还没有找到该值的时候 found 设为 false，一旦找到该值就把 found 设为 true。在一个数组中查找某个值，在文件中搜寻某员工的 ID，在一沓支票中寻找某张特定金额的支票等等的时候，都可以用 found。
- **success** 或 **ok** 用 success 或 ok 来表明一项操作是否成功。在操作失败的时候把该变量设为 false，在操作成功的时候把其设为 true。如果可以，请用一个更具体的名字代替 success，以便更具体地描述成功的含义。如果完成处理就表示这个程序执行成功，那么或许你应该用 processingComplete 来取而代之。如果找到某个值就是程序执行成功，那么你也许应该用 found。

给布尔变量赋予隐含 “真假” 含义的名字。 像 done 和 success 这样的名字是很不错的布尔变量名，因为其状态要么是 true，要么是 false；某件事完成了或者没有完成；成功或者失败。另一方面，像 status 和 sourceFile 这样的名字却是很糟的布尔变量名，因为它们没有明确的 true 或者 false。status 是 true 反映的是什么含义？它表明某件事情拥有一个状态吗？每件事都有状态。true 表明某件事情状态是 OK 吗？或者说 false 表明没有任何错误吗？对于 status 这样的名字，你什么也说不出来。

为了取得更好的效果，应该把 status 替换为类似于 error 或者 statusOK 这样的名字，同时把 sourceFile 替换为 sourceFileAvailable、sourceFileFound，或者其他能体现该变量所代表含义的名字。

有些程序员喜欢在他们写的布尔变量名前加上 Is。这样，变量名就变成了一个问题：isStatus？isError？isProcessingComplete？用 true 或 false 回答问题也就为该变量给出了取值。这种方法的优点之一是它不能用于那些模糊不清的名字：isStatus？这毫无意义。它的缺点之一是降低了简单逻辑表达式的可读性：if (isFound) 的可读性要略差于 if (Found)。

使用肯定的布尔变量名。否定的名字如 notFound、notdone 以及 notSuccessful 等较难阅读，特别是如果它们被求反： It not noFound

这样的名字应该替换为 found、done 或者 processingComplete，然后再用适当的运算符求反。如果你找到了想找的结果，那么久可以用 found 而不必写双重否定的 not notFound 了。

###为枚举类型命名

在使用枚举类型的时候，可以通过使用组前缀，如 Color_，Planet_ 或者 Month_ 来表明该类型的成员都同属于一个组。下面举一些通过前缀来确定枚举类型元素的例子：

VB 示例：为枚举类型采用前缀命名约定

```
Pubilc Enum Color
       Color_Red
       Color_Green
       Color_Blue
End Enum

Pubilc Enum Planet
       Planet_Earth
       Planet_Mars
       Planet_Venus
End Enum

Pubilc Enum Month
       Month_January
       Month_February
       ...
       Month_December
End Enum
```

与此同时，也有很多命名方法可用于确定枚举类型本身的名字(Color，Planet 或 Month)，包括全部大写或者加以前缀(e_Color，e_Planet， e_Month)。有人可能会说，枚举从本质上而言是一个用户定义类型，因此枚举名字的格式应该与其他用户定义的类型如类等相同。与之相反的一种观点认为枚举是一种类型，但它也同时是常量，因此枚举类型名字的格式应该与常量相同。本书对枚举类型采用了大小写混合的命名方式。

在有些编程语言里，枚举类型的处理很像类，枚举成员也总是被冠以枚举名字的前缀，比如 Color.Color_Red 或者 Planet.Planet_Earth。如果你正在使用这样的编程语言，那么重复上述前缀的意义就不大了，因此你可以吧枚举类型自身的名字作为前缀，并把上述名字简化为 Color.Red 和 Planet.Earth。

###为常量命名

在具名常量时，应该根据该常量所表示的含义，而不是该常量所具有的数值为该抽象事物命名。FIVE 是个很糟的常量名（不论它所代表的值是否为 5.0）。CYCLES_NEEDED 是一个不错的名字。CYCLES_NEEDED 可以等于 5.0 或者 6.0。而 FIVE = 6.0 就显得太可笑了。出于同样原因，BAKERS_DOZEN 就是个很糟的常量名，而 DONUTS_MAX 则不错。