

命名约定

本书采用的命名约定和 Microsoft 所使用的“匈牙利式”命名约定差不多。该约定是由生于匈牙利布达佩斯的 Charles Simonyi 开发的，它通过在数据和函数名中加入额外的信息以增进程序员对程序的理解。例如：

```
char ch;           /* 所有的字符变量均以 ch 开始 */
byte b;            /* 所有的字节均冠以 b */
long l;            /* 所有的长字均冠以 l */
```

对于指向某个数据类型的指针，可以先象上面那样建立一个有类型的名字，然后给该名字加上前缀字母 P：

```
char* pch;          /* 指向 ch 的指针以 p 开始 */
byte* pb;           /* 同理 */
long* pl;
void* pv;           /* 特意显用的空指针 */
char** ppch;        /* 指向字符指针的指针 */
byte** ppb;         /* 指向字节指针的指针 */
```

匈牙利式名字通常不那么好念，但在代码中读到它们时，确实可以从中得到许多的信息。例如，当你眼看到某个函数里有一个名为 pch 的变量时，不用查看声明就立即知道它是一个指向字符的指针。

为了使匈牙利式名字的描述性更强。或者要区分两个变量名，可以在相应类型派生出的基本名字之后加上一个以大写字母开头的“标签”。例如，strcpy 函数有两个字符指针参数：一个是源指针，另一个是目的指针。使用匈牙利式命名约定，其相应的原型是：

```
char* strcpy(char* pchTo, char* pchFrom); /* 原型 */
```

在上面的例子中，两个字符指针有一个共同的特点——都指向以 0 为结尾的 C 的字符串。因此在本书中，每当用字符指针指向字符串时，我们就用一个更有意义的名字 str 来表示。因此，上述 strcpy 的原型则为：

```
char* strcpy(char* strTo, char* strFrom) /* 原型 */
```

本书用到另一个类型是 ANSI 标准中的类型 size_t。下面给出该类型的一些典型用法：

```
size_t sizeNew, sizeOld; /* 原型 */
void* malloc(size_t size); /* 原型 */
void* realloc(void* pv, size_t sizeNew); /* 原型 */
```

函数和数组的命名遵循同样的约定，名字由相应的返回类型名开始，后跟一个描述的标签。例如：

```
ch = chLastKeyPressed; /* 由变量得一字符 */
ch = chInputBuffer[]; /* 由数组得一字符 */
ch = chReadKeyboard; /* 由函数得一字符 */
```

如果利用匈牙利式命名方法，mall~和 reali~可以写成如下形式：

```
void* pvNewBlock(size_t size);           /* 原型 */  
void* pvResizeBlock(void* pv, size_t sizeNew); /* 原型 */
```

由于匈牙利式命名方法旨在增进程序员对程序的理解，所以大多数匈牙利式名字的长度都要超过 ANSI 严格规定 6 个字母的限制。这就不妙，除非所用的系统是几十年前设计的系统，否则这 6 个字母的限制只当是历史的遗迹。

以上内容基本上没有涉及到匈牙利式命名约定的细节，所介绍的都是读者理解本书中所用变量和函数名称意义的必需内容。如果读者对匈牙利式命名约定的详细内容感兴趣，可以参考本书末尾参考文献部分列出的 Simonyi 的博士论文。