# 7.8 函数与array对象

在C++中，类对象是基于结构的，因此结构编程方面的有些考虑因素也适用于类。例如，可按值将对象传递给函数，在这种情况下，函数处理的是原始对象的副本。另外，也可传递指向对象的指针，这让函数能够操作原始对象。下面来看一个使用C++11模板类array的例子。

假设您要使用一个array对象来存储一年四个季度的开支：

std：array<double,4> expenses;

要使用array类，需要包含头文件array，而名称array位于命名空间std中，如果函数来显示expenses的内容，可按值传递expenses：

show(expenses);

但如果函数要修改对象expenses，则需将该对象的地址传递给函数：

fill(&expenses);

这与程序清单7.13处理结构时使用的方法相同。

如何声明这两个函数呢？expenses的类型为array<double,4>，因此必须在函数原型中指定这种类型：

void show(std::array<double,4> da); // da是一个对象

void fill(std::array<double,4) \*pa); // pa是指向对象的指针

这些考虑因素是这个示例程序的核心。该程序还包含其他一些功能。首先，它用符号常量替换了4：

const int Seasons = 4;

其次，它使用了一个const array对象，该对象包含4个string对象，用于表示几个季度：

const std::array<std::string,Seasons> Snames = {”Sprrng”, ”Summer”,”Fall”,”Winter”};

请注意，模板array并非只能存储基本数据类型，它还可存储类对象。程序清单7.15列出了该程序的完整代码。

**程序清单7.15 arrobj.cpp**

//arrobj.cpp -- 使用array对象的函数

#include <iostream>

#include <array>

#include <string>

const int Seasons = 4;

const std::array<std::string, Seasons> Snames =

 {"Spring", "Summer", "Fall", "Winter"};

void fill(std::array<double, Seasons> \* pa);

void show(std::array<double, Seasons> da);

int main()

{

 std::array<double, 4> expenses;

 fill(&expenses);

 show(expenses);

 std::cin.get();

 std::cin.get();

 return 0;

}

void fill(std::array<double, Seasons> \* pa)

{

 for (int i = 0; i < Seasons; i++)

 {

 std::cout << "Enter " << Snames[i] << " expenses: ";

 std::cin >> (\*pa)[i];

 }

}

void show(std::array<double, Seasons> da)

{

 double total = 0.0;

 std::cout << "\nEXPENSES\n";

 for (int i = 0; i < Seasons; i++)

 {

 std::cout << Snames[i] << ": $" << da[i] << '\n';

 total += da[i];

 }

 std::cout << "Total: $" << total << '\n';

}

下面是该程序的运行情况：

Enter Spring expenses: 212

Enter Summer expenses: 256

Enter Fall expenses: 208

Enter Winter expenses: 244

EXPENSES

Spring: $212

Summer: $256

Fall:$208

Winter: $244

Total: $920

## 程序说明

由于const array对象Snames是在所有函数之前声明的，因此可在后面的任何函数定义中使用它。与const Seasons一样，Snames也由整个源代码文件共享。这个程序没有使用编译指令using，因此必须使用std::限定array和string。为简化程序，并将重点放在函数可如何使用对象上，函数fill()没有检查输入是否有效。

函数fill()和show()都有缺点。函数show()存在的问题是，expenses存储了四个double值，而创建一个新对象并将expenses的值复制到其中的效率太低。如果修改该程序，使其处理每月甚至每日的开支，这种问题将更严重。

函数fill()使用指针来直接处理原始对象，这避免了上述效率低下的问题，但代价是代码看起来更复杂：

fill(&expenses); // 别忘了&

…

cin >> (\*pa)[i];

在最后一条语句中，pa是一个指向array<double，4>对象的指针，因此\*pa为这种对象，而(\*pa)[i]是该对象的一个元素。由于运算符优先级的影响，其中的括号必不可少。这里的逻辑很简单，但增加了犯错的机会。

使用第8章将讨论的引用可解决效率和表示法两方面的问题。