Functions and `array` Objects

Class objects in C++ are based on structures, so some of the same programming considerations that apply to structures also apply to classes. For example, you can pass an object by value to a function, in which case the function acts on a copy of the original object. Alternatively, you can pass a pointer to an object, which allows the function to act on the original object. Let's look at an example using the C++11 `array` template class.

Suppose we have an `array` object intended to hold expense figures for each of the four seasons of the year:

```
std::array<double, 4> expenses;
```

(Recall that using the `array` class requires the `array` header file and that the name `array` is part of the `std` namespace.) If we want a function to display the contents of `expenses`, we can pass `expenses` by value:

```
show(expenses);
```

But if we want a function that modifies the `expenses` object, we need to pass the address of the object to the function:

```
fill(&expenses);
```

(The next chapter discusses an alternative approach, using references.) This is the same approach that Listing 7.13 used for structures.

How can we declare these two functions? The type of `expenses` is `array<double, 4>`, so that's what must appear in the prototypes:

```
void show(std::array<double, 4> da);   // da an object
void fill(std::array<double, 4> * pa); // pa a pointer to an object
```

These considerations form the core of the sample program. The program adds a few more features. First, it replaces `4` with a symbolic constant:

```
const int Seasons = 4;
```

Second, it adds a `const array` object containing four `string` objects representing the four seasons:

```
const std::array<std::string, Seasons> Snames =
    {"Spring", "Summer", "Fall", "Winter"};
```

Note that the `array` template is not limited to holding the basic data types; it can use class types too. Listing 7.15 presents the program in full.

Listing 7.15 **arrobj.cpp**

```
//arrobj.cpp -- functions with array objects (C++11)
#include <iostream>
#include <array>
#include <string>
// constant data
```

```
const int Seasons = 4;
const std::array<std::string, Seasons> Snames =
    {"Spring", "Summer", "Fall", "Winter"};

// function to modify array object
void fill(std::array<double, Seasons> * pa);
// function that uses array object without modifying it
void show(std::array<double, Seasons> da);

int main()
{
    std::array<double, Seasons> expenses;
    fill(&expenses);
    show(expenses);
    return 0;
}

void fill(std::array<double, Seasons> * pa)
{
    using namespace std;
    for (int i = 0; i < Seasons; i++)
    {
        cout << "Enter " << Snames[i] << " expenses: ";
        cin >> (*pa)[i];
    }
}

void show(std::array<double, Seasons> da)
{
    using namespace std;
    double total = 0.0;
    cout << "\nEXPENSES\n";
    for (int i = 0; i < Seasons; i++)
    {
        cout << Snames[i] << ": $" << da[i] << endl;
        total += da[i];
    }
    cout << "Total Expenses: $" << total << endl;
}
```

Here's a sample run:

```
Enter Spring expenses: 212
Enter Summer expenses: 256
Enter Fall expenses: 208
Enter Winter expenses: 244
```