array name as an argument. And to make the function general so that it is not restricted to an array of a particular size, you pass the size of the array. The only new ingredient here is that you have to declare that one of the formal arguments is an array name. Let's see what that and the rest of the function heading look like:

```c++
int sum_arr(int arr[], int n) // arr = array name, n = size
```

This looks plausible. The brackets seem to indicate that \texttt{arr} is an array, and the fact that the brackets are empty seems to indicate you can use the function with an array of any size. But things are not always what they seem: \texttt{arr} is not really an array; it's a pointer! The good news is that you can write the rest of the function just as if \texttt{arr} were an array. First, let's see that this approach works, and then let's look into why it works.

Listing 7.5 illustrates using a pointer as if it were an array name. The program initializes the array to some values and uses the \texttt{sum\_arr()} function to calculate the sum. Note that the \texttt{sum\_arr()} function uses \texttt{arr} as if it were an array name.

\begin{verbatim}
Listing 7.5 arrfun1.cpp

// arrfun1.cpp -- functions with an array argument
#include <iostream>
using namespace std;
const int ArSize = 8;
int sum_arr(int arr[], int n); // prototype
int main()
{
    int cookies[ArSize] = {1,2,4,8,16,32,64,128};
    // some systems require preceding int with static to // enable array initialization
    int sum = sum_arr(cookies, ArSize);
    cout << "Total cookies eaten: " << sum << "\n";
    return 0;
}

// return the sum of an integer array
int sum_arr(int arr[], int n)
\end{verbatim}