This chapter begins with a typical data collection problem to illustrate the utility of and Fortran syntax for defining and using arrays. Topics covered include one-dimensional and multidimensional arrays, arrays as parameters in functions and subroutines, allocatable arrays, strings as “arrays” of characters, user-defined record types, and arrays of records.

8.1 Arrays in Structured Programming

Up to now, this text has discussed only data types for which a variable name corresponds to a single value and a single location in memory.\(^1\) It is possible, and very desirable in structured programming, to define other kinds of data structures that lead to more convenient and compact ways of organizing information in a program. The most important user-defined data structure is the array. Basically, an array is a collection of related values organized under a single name. In this section, we will develop the basic concepts of arrays by posing a specific data management problem and examining how best to organize the information required to solve that problem.

Suppose you are conducting an experiment to monitor the concentration of tropospheric ozone, a federally regulated air pollutant. You have in place equipment that produces one measurement per hour for 24 hours. You would like to store these measurements and then write a program to process the data. How should your program handle this task? One way would be to associate each ozone measurement with its own variable name:

\[
\begin{align*}
\text{Ozone1} \\
\text{Ozone2} \\
\text{Ozone3} \\
\text{Ozone4} \\
\ldots \\
\text{Ozone23} \\
\text{Ozone24}
\end{align*}
\]

\(^1\)This simple concept is adequate as long as we recognize that a single memory “location” will contain as many bytes as required to hold the variable, based on its data type declaration.
This seems a little awkward, but it will become much worse if you decide that what you really need is hourly measurements for 10 days. Suddenly you're faced with creating another 216 variable names!

Fortunately, there's an easier way: define a single variable name—Ozone—and an indexing system that can be used to address all the ozone measurements under this single variable name. Symbolically, each measurement could be addressed like this:

```
Ozone(1)
Ozone(2)
Ozone(3)
Ozone(4)
...
Ozone(23)
Ozone(24)
```

The interpretation of this system is the obvious one: Ozone(1) is the measurement at the first hour, Ozone(2) is the measurement at the second hour, and so forth. If you need more measurements, all you have to do is increase the value of the largest index, from 24 to 240, for example:

```
Ozone(1)
Ozone(2)
Ozone(3)
Ozone(4)
...
Ozone(239)
Ozone(240)
```

When it's translated into a programming language, this kind of data representation is called an array. The name of the array will be something obvious and descriptive, like Ozone. The number in parentheses is the **array index**. The value associated with each index is called an **array element**. The number of elements is called the **extent** (or size) of the array. For an array holding 24 ozone measurements per day for 10 days, the extent would be 240. For the above example, the notation Ozone(26) will refer to the 26th element—the second measurement on the second day—in the array named Ozone. This notation seems perfectly straightforward, and as we will see, it is just the notation that Fortran uses.

One-dimensional arrays are often associated with vectors, in the physical sense, or with **vector data** in a somewhat more generalized sense, as opposed to **scalar data**. This is a distinction that should be familiar from an introductory physics course. To cite some examples, the speed of a moving object is represented by a single number and is a scalar quantity. The velocity of a moving object is a vector quantity that describes both speed and direction with...