Chapter 6

Pointers and Arrays

It is difficult to imagine performing many scientific calculations without the ability to manipulate arrays of one or more dimensions. Arrays are needed for vectors, matrices and the convenient storage of related data. In this chapter we introduce a simple picture of the way memory is organized, together with powerful methods for using contiguous areas of memory.

6.1 Memory, Addressing and Pointers

The memory of a computer can be regarded as a collection of labelled storage locations, as shown in Figure 6.1. In our discussions we assume that we can access these memory locations in any order. A very simplified picture is to visualize memory as a linear sequence of storage locations, one byte in size, which are labelled 1, 2, 3 ... etc. A particular label is known as the address of the corresponding memory element. The two operations of interest are to read what is stored at a particular address and to write data to the memory labelled by an address.

6.1.1 'address-of' Operator

In C++ the address of any variable can always be found by preceding the variable with the address-of operator, denoted, &. The address-of operator has the same precedence and right to left associativity as other unary operators. Several new operators are defined during the course of this chapter; their precedence and associativity are given in Appendix C. In the present context, the & operator simply means ‘return the address of the variable to the right’. The following example illustrates the difference between values stored by variables of various types and their corresponding addresses:

```c
int i = 1;
double x = 3.0;
```
float z = 4.0;
cout << "Address of i is " << &i <<
    " whereas the value of i is " << i << "\n";
cout << "Address of x is " << &x <<
    " whereas the value of x is " << x << "\n";
cout << "Address of z is " << &z <<
    " whereas the value of z is " << z << "\n";

Exercise:

Run the above code on your system. You may notice that the memory
locations given by the above code differ by the size of the various
types. (Whether or not this occurs depends on how the compiler
allocates storage.)

<table>
<thead>
<tr>
<th>Address Values in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

There are a few things that we cannot do with the address-of operator. It is
illegal to take the address of a constant:

&10; // WRONG: cannot take the address of a constant.
&3.142; // WRONG: cannot take the address of a constant.

It is also illegal to take the address of an expression:

float x = 302.8;
&(x + 73.6); // WRONG: cannot take the address
              // of expressions.