Arrays and Loops

Just as with `vector`, the real advantage of arrays is that you can automate the processing of a collection of related elements, like the grades for all the students in a class.

However, because arrays are lower-level structures, processing them is not quite as convenient. There are several ways to use loops to traverse an array.

1. Calculate the **number of elements in the array** and use that as a limit on a traditional counter-controlled ***`for`*** or ***`while`*** loop.
2. Use a **sentinel value** stored in the array to mark its end.
3. Use a pair of pointers: one to the first element in the array, and one to the address right past the end of the array. These are called **iterator-based** loops.
4. Use the C++ 11 **range-based** ***`for`*** loop.

Inside a function, **only the first three are meaningful**. You **cannot** use the range-based for loop on an array name after it has decayed to a pointer.

The range-based for Loop

You may, however, use the **range-based** ***`for`*** loop on arrays, provided that the array definition is **in scope**. Here's an example:

```
int a[] = {...};
for (int e : a)
{
  cout << e << " ";
}
```